





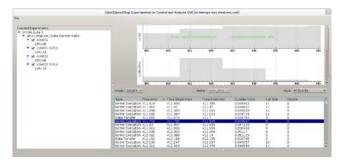
# Open | SpeedShop™ Performance with Open | SpeedShop



## NASA Open | SpeedShop Update / Training

Jim Galarowicz: Argo Navis, Krell Institute
Greg Schultz, Argo Navis
Don Maghrak, Krell Institute
William Hachfeld, Argo Navis, Krell Institute









## **Webinar Preparation Underway**

If you cannot hear meeting room activity on your computer:

- 1. Be sure your computer audio volume is high enough to hear
- 2. If there's still a problem, use the "Chat Box" facility of Webex to request a telephone callback. (The Chat box is reachable from the green tab [Viewing How Code for Screen.)

## Instructions for Participants





#### How to hear the audio (in order of preference):

- Use your computer audio to follow the training
  - This should work with Windows and Mac, but maybe not with Linux
- 2. Dial 650-479-3208

#### Participants not in the meeting room at NAS have been muted

This should reduce cross-talk and provide everyone with better audio

#### Questions are welcome during the presentation:

- Ask a question in the Chat facility
  - The host will be monitoring and will relay your question to the speaker
- ➤ If the question is too complicated to ask in text, call 650-479-3208 and ask the host to unmute you in order for you to communicate by phone.

## Presenters and Extended Team





- **❖ Jim Galarowicz, Argo Navis, Krell Institute**
- Greg Schultz, Argo Navis
- Don Maghrak, Krell Institute
- William Hachfeld, Argo Navis, Krell Institute



#### **Open|SpeedShop extended team:**

- Patrick Romero: Krell Institute
- Jennifer Green, David Montoya, Mike Mason, David Shrader: LANL
- ❖ Martin Schulz, Matt Legendre and Chris Chambreau: LLNL
- Mahesh Rajan, Doug Pase, Anthony Agelastos: SNL
- Dyninst group (Bart Miller: UW & Jeff Hollingsworth: UMD)
- ❖ Phil Roth, Mike Brim: ORNL
- Ciera Jaspan: CMU



## Outline





### Section 1: Introduction to Open | SpeedShop tools

- How to use Open | SpeedShop to gather and display
- Overview of performance experiments
  - Sampling Experiments and Tracing Experiments
- How to compare performance data for different application runs

## **Section 2: New Functionality/Experiments**

- Memory (ossmem) experiment
- OpenMP augmentation and OMPTP (ossomptp) experiment
- > POSIX threads (osspthreads) experiment
- Lightweight experiments (ossiop, ossmpip)
- NVIDIA CUDA tracing experiment (osscuda)

## **Section 3: Roadmap / Future Plans**

## **Supplemental Information**

> Command Line Interface (CLI) tutorial and examples

## NASA Open | SpeedShop Availability





#### **Pleaides platform:**

- module use /home4/jgalarow/privatemodules
- Module names:
  - module load openspeedshop (defaults to mpt)
  - > module load openspeedshop.mpt
  - > module load openspeedshop.intelmpi
  - > module load openspeedshop.mvapich2
  - > module load openspeedshop.openmpi

#### **KNL** cluster platform:

- module use /u/jgalarow/privatemodules
- Module names:
  - module load openspeedshop (defaults to mpt)
  - module load openspeedshop.mpt
  - > module load openspeedshop.intelmpi

For mpi\* experiments use the module file that corresponds to the MPI implementation your application was built with.







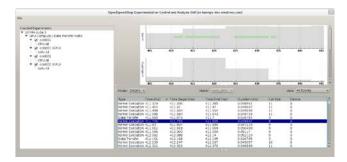
## Open | SpeedShop™ Performance with Open | SpeedShop



# NASA Open | SpeedShop Update / Training

## Section 1 Introduction into Tools and Open|SpeedShop







## Open | SpeedShop Tool Set





## Open Source Performance Analysis Tool Framework

- > Most common performance analysis steps all in one tool
- Combines tracing and sampling techniques
- > No need to recompile the application being monitored.
- > Gathers and displays several types of performance information
- > Maps performance data information to application source code

#### Flexible and Easy to use

User access through:
GUI, Command Line, Python Scripting, convenience scripts

#### Scalable Data Collection

- > Instrumentation of *unmodified application binaries*
- > New option for *hierarchical online data aggregation*

### Supports a wide range of systems

- > Extensively used and tested on a variety of *Linux clusters*
- > Cray, Blue Gene, ARM, Intel MIC, GPU support

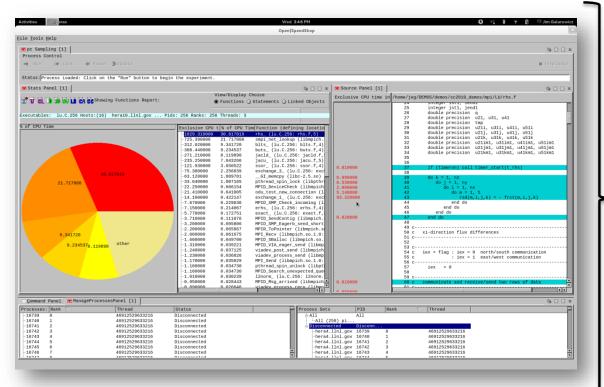
## Open | SpeedShop Workflow - default

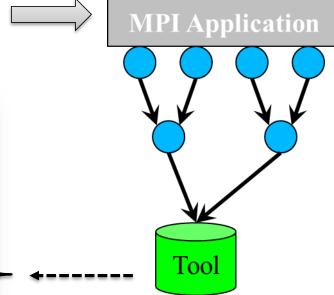




mpiexec\_mpt -np 4 smg2000 -n 65 65 65

osspcsamp "mpiexec\_mpt -np 4 smg2000 -n 65 65 65"





http://www.openspeedshop.org/

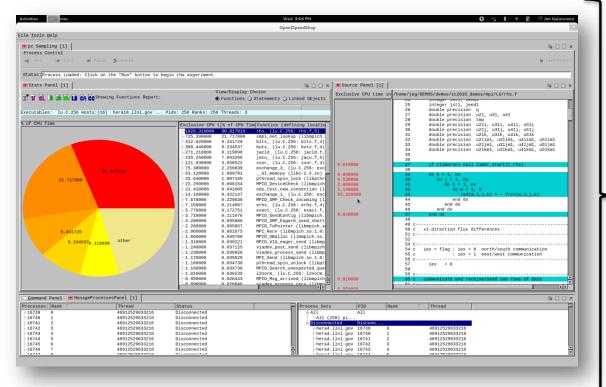
## Open | SpeedShop Workflow – optional

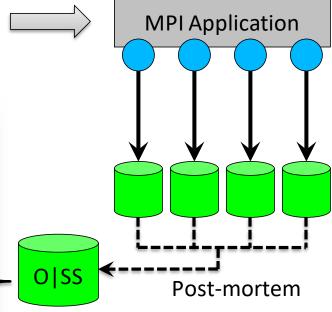




mpiexec\_mpt -np 4 smg2000 -n 65 65 65

osspcsamp -- offline "mpiexec\_mpt -np 4 smg2000 -n 65 65 65"





http://www.openspeedshop.org/

#### Alternative Interfaces





### Scripting language

- > Immediate command interface
- ➤ O|SS interactive command line (CLI)
  - openss -cli

```
Experiment Commands
expView
expCompare
expStatus
```

```
List Commands
list -v exp
list -v hosts
```

#### 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)
```

## **Central Concept: Experiments**





## Users pick experiments:

- What to measure and from which sources?
- > How to select, view, and analyze the resulting data?

#### Two main classes:

- Statistical Sampling
  - Periodically interrupt execution and record location
  - Useful to get an overview
  - Low and uniform overhead
- Event Tracing
  - Gather and store individual application events
  - Provides detailed per event information
  - Can lead to huge data volumes

## O|SS can be extended with additional experiments

## Sampling Experiments in O|SS





#### PC Sampling (pcsamp)

- Record PC repeatedly at user defined time interval
- 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 (hwc, hwctime, hwcsamp)

- > Provides profile of hardware counter events like cache & TLB misses
- > hwcsamp:
  - Periodically sample to capture profile of the code against the chosen counter
  - Default events are PAPI\_TOT\_INS and PAPI\_TOT\_CYC
- hwc, hwctime:
  - Sample a hardware counter till a certain number of events (called threshold) is recorded and get Call Stack
  - Default event is PAPI TOT CYC overflows

## Tracing Experiments in O|SS





## Input/Output Tracing (io, iot, iop)

- > Record invocation of all POSIX I/O events
- > Provides aggregate and individual timings
- Store function arguments and return code for each call (iot)
- Lightweight I/O profiling because not tracking individual call details (iop)

## MPI Tracing (mpi, mpit, mpip)

- > Record invocation of all MPI routines
- Provides aggregate and individual timings
- Store function arguments and return code for each call (mpit)
- Lightweight MPI profiling because not tracking individual call details (mpip)

## Memory Tracing (mem)

- > Record invocation of key memory related function call events
- Provides aggregate and individual rank, thread, or process timings

## Tracing Experiments in O|SS





### CUDA NVIDIA GPU Event Tracing (cuda)

- > Record CUDA events, provides timeline and event timings
- > 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.

## POSIX thread tracing (pthreads)

- > Record invocation of all POSIX thread events
- Provides aggregate and individual rank, thread, or process timings

## OpenMP specific profiling/tracing (omptp)

Report task idle, barrier, and barrier wait times per OpenMP thread and attribute those times to the OpenMP parallel regions.

## How to Run a First Experiment in O|SS?





## Picking the experiment

- What do I want to measure?
- We will start with pcsamp to get a first overview

### 2. Launching the application

- How do I control my application under O|SS?
- Enclose how you normally run your application in quotes
- osspcsamp "mpirun –np 4 smg2000 –n 65 65 65"

### 3. Storing the results

- O/SS will create a database
- Name: smg2000-pcsamp.openss

## 4. Exploring the gathered data

- How do I interpret the data?
- O|SS will print a default report (offline version only)
- Open the GUI to analyze data in detail (run: "openss")

## Example Run with Output (1 of 2)





## ❖ osspcsamp "mpirun –np 4 smg2000 –n 65 65 65"

```
Bash> osspcsamp "mpirun -np 4 ./smg2000 -n 65 65 65"
[openss]: pcsamp experiment using the pcsamp experiment default sampling rate: "100".
[openss]: Using OPENSS PREFIX installed in /opt/ossoffv2.1u4
[openss]: Setting up offline raw data directory in /opt/shared/offline-oss
[openss]: Running offline pcsamp experiment using the command:
"mpirun -np 4 /opt/ossoffv2.1u4/bin/ossrun"./smg2000 -n 65 65 65" pcsamp"
Running with these driver parameters:
(nx, ny, nz) = (65, 65, 65)
     <SMG native output>
Final Relative Residual Norm = 1.774415e-07
[openss]: Converting raw data from /opt/shared/offline-oss into temp file X.O.openss
Processing raw data for smg2000
Processing processes and threads ...
Processing performance data ...
Processing functions and statements ...
Resolving symbols for /home/jeg/DEMOS/workshop_demos/mpi/smg2000/test/smg2000
```

## Example Run with Output (2 of 2)





## ❖ osspcsamp "mpirun –np 4 smg2000 –n 65 65 65"

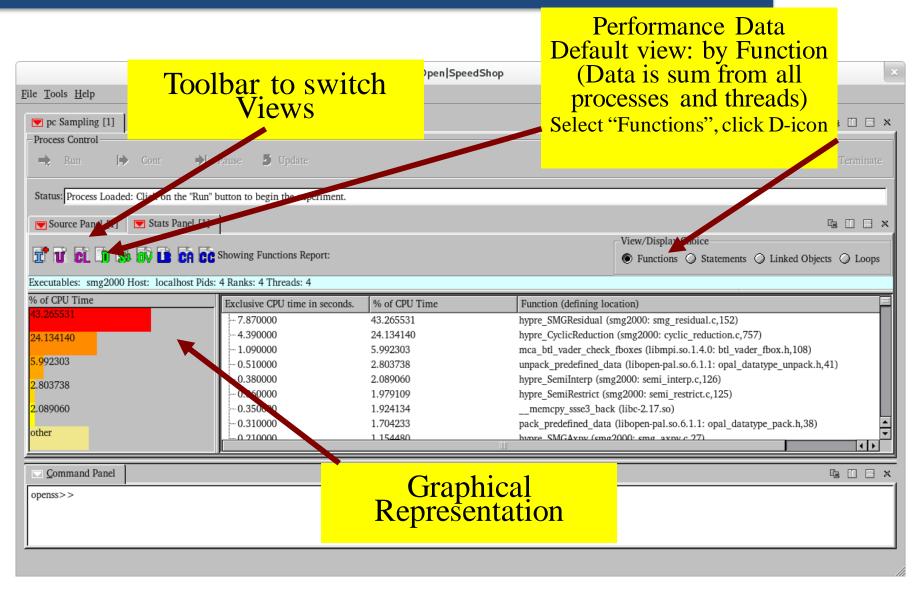
```
[openss]: Restoring and displaying default view for:
     /home/jeg/DEMOS/workshop_demos/mpi/smg2000/test/smg2000-pcsamp.openss
[openss]: The restored experiment identifier is: -x 1
Exclusive CPU time
                     % of CPU Time Function (defining location)
   in seconds.
  7.870000 43.265531 hypre_SMGResidual (smg2000: smg_residual.c,152)
  4.390000 24.134140 hypre CyclicReduction (smg2000: cyclic reduction.c,757)
  1.090000 5.992303 mca_btl_vader_check_fboxes (libmpi.so.1.4.0: btl_vader_fbox.h,108)
  0.510000 2.803738 unpack predefined data (libopen-pal.so.6.1.1: opal_datatype_unpack.h,41)
 0.380000 2.089060 hypre_SemiInterp (smg2000: semi_interp.c,126)
  0.360000 1.979109 hypre SemiRestrict (smg2000: semi restrict.c, 125)
  0.350000 1.924134 memcpy ssse3 back (libc-2.17.so)
  0.310000 1.704233 pack predefined data (libopen-pal.so.6.1.1: opal_datatype_pack.h,38)
 0.210000 1.154480 hypre_SMGAxpy (smg2000: smg_axpy.c,27)
 0.140000  0.769654 hypre_StructAxpy (smg2000: struct_axpy.c,25)
  0.110000 0.604728 hypre SMGSetStructVectorConstantValues (smg2000: smg.c,379)
```

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

## **Default Output Report View**



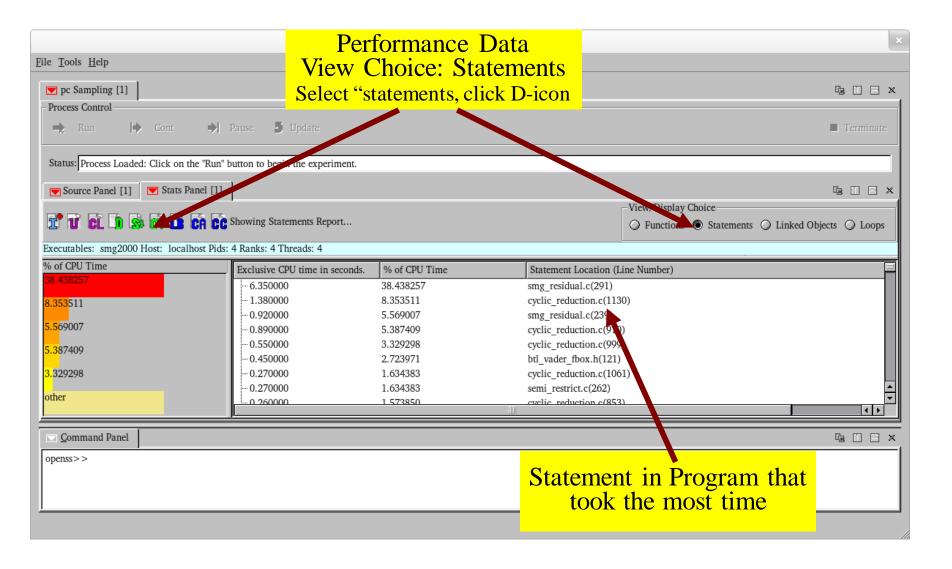




## Statement Report Output View



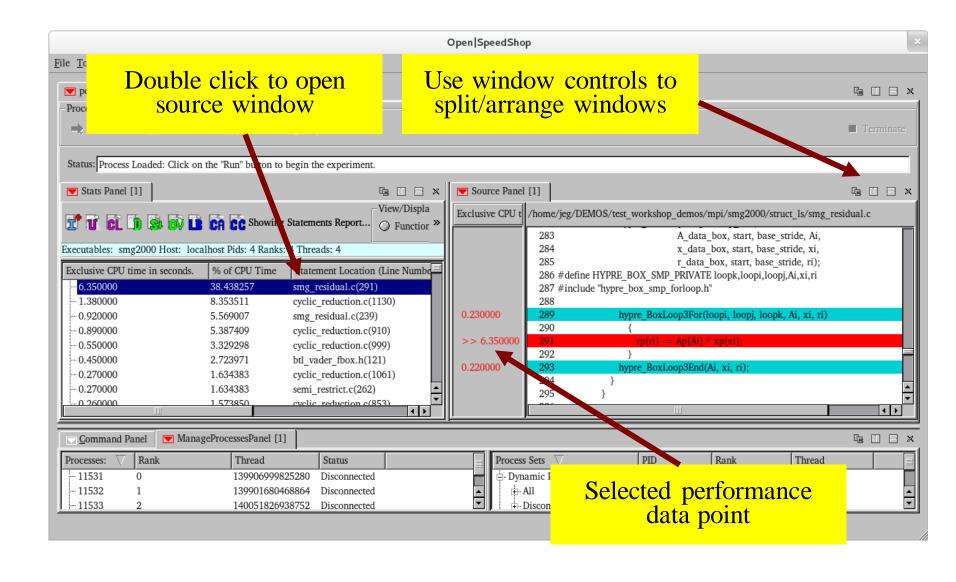




### Associate Source & Performance Data



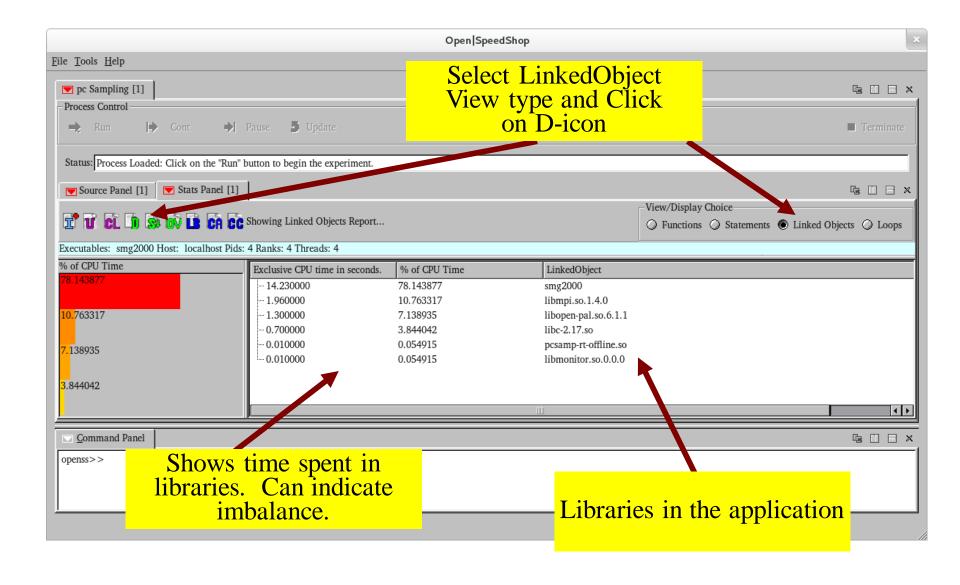




## Library (LinkedObject) View



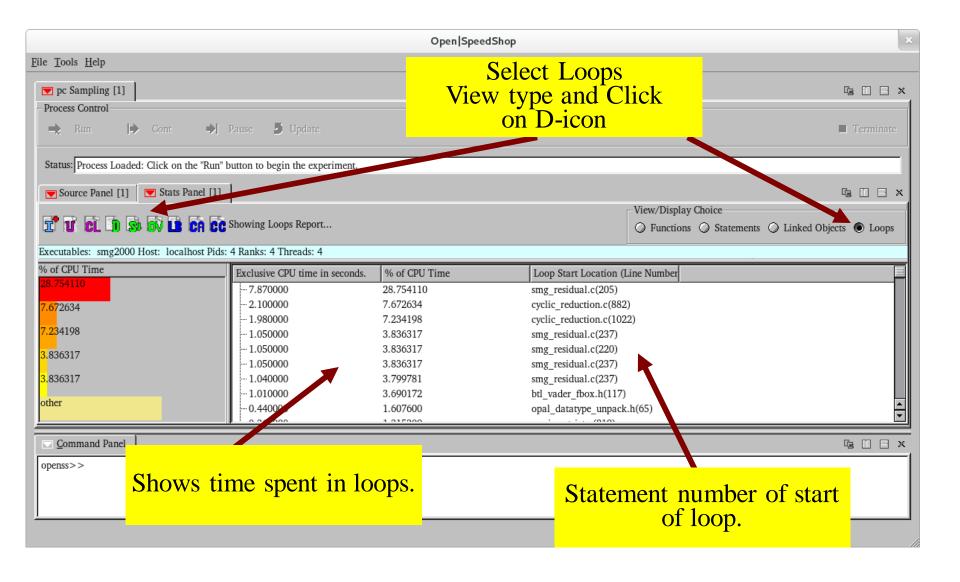




## **Loop View**







## First Experiment Run: Summary





- Place the way you run your application normally in quotes and pass it as an argument to osspcsamp, or any of the other experiment convenience scripts: ossio, ossmpi, etc.
  - osspcsamp "mpiexec\_mpt -np 64 ./mpi\_application app\_args"
- Open|SpeedShop sends a summary profile to stdout
- Open|SpeedShop creates a database file
- Display alternative views of the data with the GUI via:
  - openss –f <database file>
- Display alternative views of the data with the CLI via:
  - openss –cli –f <database file>
- On clusters, need to set OPENSS\_RAWDATA\_DIR
  - > Should point to a directory in a shared file system
  - Usually set/handled in a module or dotkit file.
- Start with pcsamp for overview of performance
- Then, focus on performance issues with other experiments

## **Identifying Critical Regions**





#### Flat Profile Overview

## Profiles show computationally intensive code regions

First views: Time spent per functions or per statements

#### Questions:

- > Are those functions/statements expected?
- Do they match the computational kernels?
- Any runtime functions taking a lot of time?

### Identify bottleneck components

- View the profile aggregated by shared objects (LinkedObject view)
- Correct/expected modules?
- > Impact of support and runtime libraries







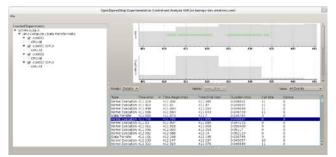
## Open | SpeedShop™ Performance with Open | SpeedShop



## NASA Open | SpeedShop Update/Training

## Call Path Profiling (usertime)







## Call stack profiling





## Call Stack Profiling

- Take a sample: address inside a function
- > Call stack: series of program counter addresses (PCs)
- Unwinding the stack is walking through those addresses and recording that information for symbol resolution later.
- Leaf function is at the end of the call stack list

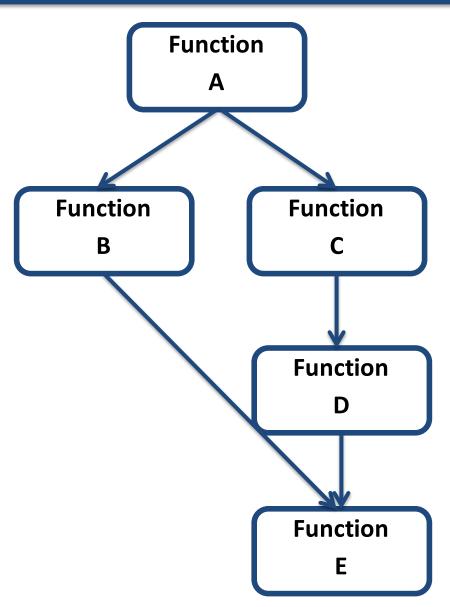
## Open|SpeedShop: experiment called usertime

- > Time spent inside a routine vs. its children
- Time spent along call paths in the application
- > Key view: butterfly

## Adding Context through Stack Traces







## Missing information in flat profiles

- Distinguish routines called from multiple callers
- Understand the call invocation history
- Context for performance data

#### Critical technique: Stack traces

- Gather stack trace for each performance sample
- Aggregate only samples with equal trace

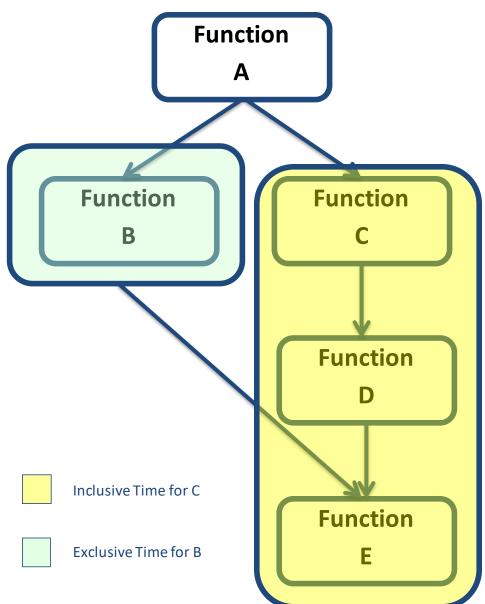
#### User perspective:

- Butterfly views (caller/callee relationships)
- > Hot call paths
  - Paths through application that take most time

## Inclusive vs. Exclusive Timing







## Stack traces enable calculation of inclusive/exclusive times

- Time spent inside a function only (exclusive)
  - See: Function B
- Time spent inside a function and its children (inclusive)
  - See Function C and children

#### Implementation similar to flat profiles

- Sample PC information
- Additionally collect call stack information at every sample

#### Tradeoffs

- Pro: Obtain additional context information
- Con: Higher overhead/lower sampling rate

## Interpreting Call Context Data





#### Inclusive versus exclusive times

- > If similar: child executions are insignificant
  - May not be useful to profile below this layer
- > If inclusive time significantly greater than exclusive time:
  - Focus attention to the execution times of the children

### Hotpath analysis

- Which paths takes the most time?
- > Path time might be ok & expected, but could point to a problem

## Butterfly analysis (similar to gprof)

- > Could be done on "suspicious" functions
  - Functions with large execution time
  - Functions with large difference between implicit and explicit time
  - Functions of interest
  - Functions that "take unexpectedly long"
  - ...
- Shows split of time in callees and callers

## In/Exclusive Time in O|SS: Usertime





### **Basic syntax:**

ossusertime "how you run your executable normally"

#### **Examples:**

ossusertime "smg2000 –n 50 50 50" low ossusertime "smg2000 –n 50 50 50" low

#### Parameters

Sampling frequency (samples per second)
Alternative parameter: high (70) | low (18) | default (35)

Recommendation: compile code with -g to get statements!

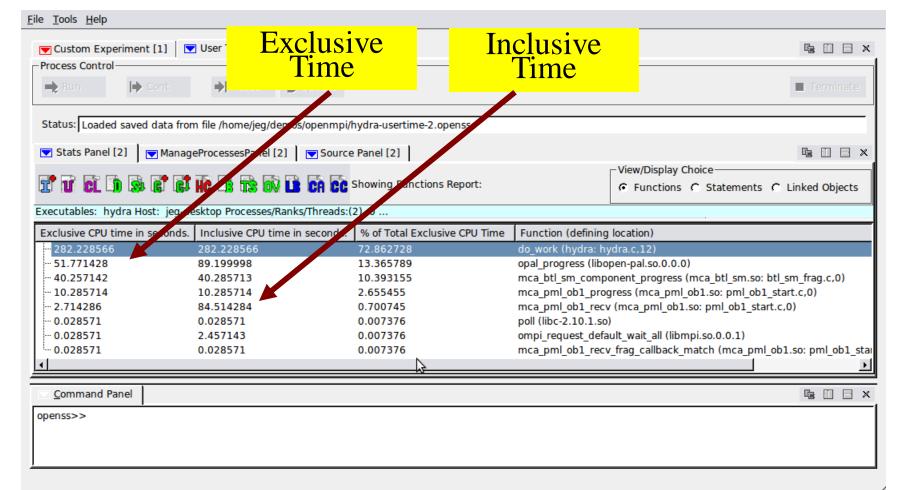
## Reading Inclusive/Exclusive Timings





#### Default View

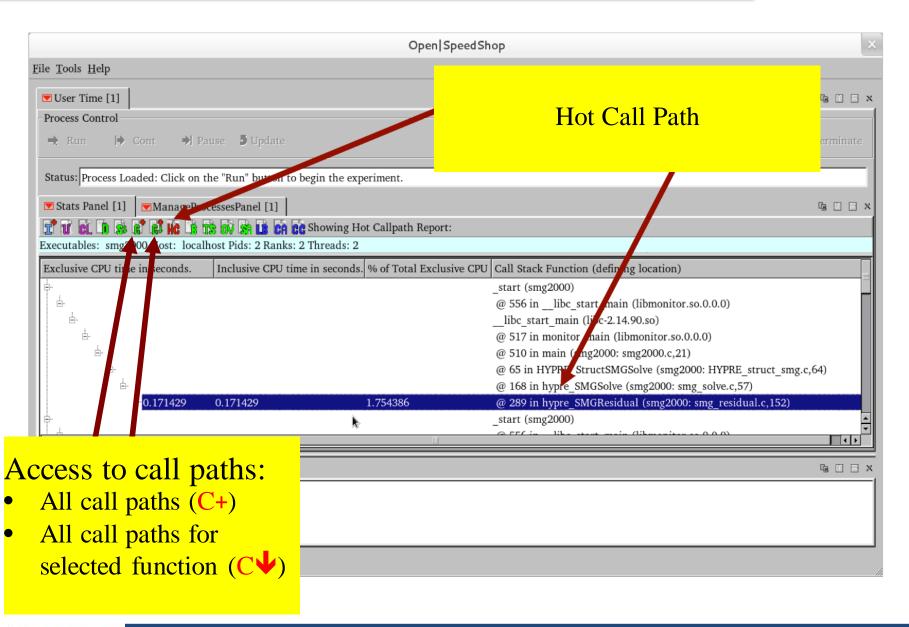
- > Similar to pcsamp view from first example
- Calculates inclusive versus exclusive times



### Stack Trace Views: Hot Call Path





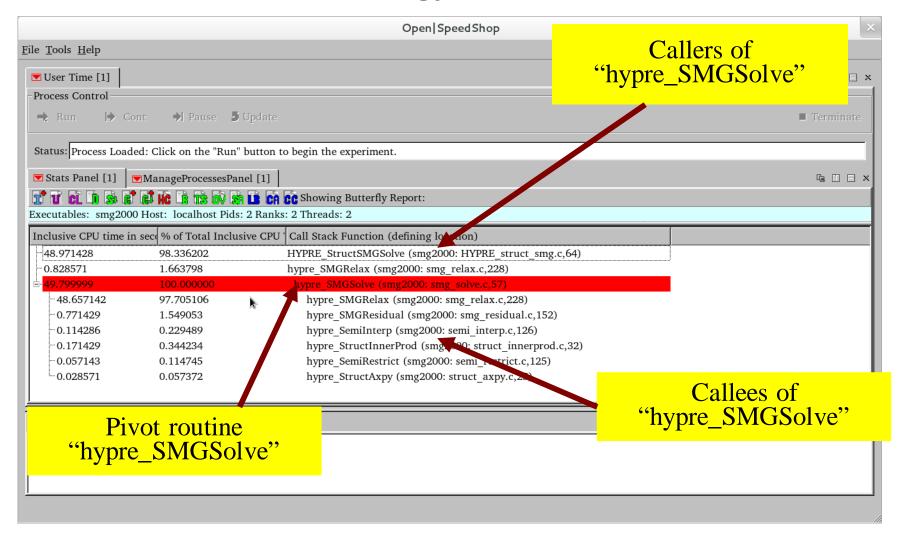


## Stack Trace Views: Butterfly View





### Similar to well known "gprof" tool



## Demonstration: Call path profiling





- Usertime experiment related application exercise
- Call path profiling exercises can be found in these directories:
  - > \$HOME/exercises/seq\_smg2000
  - > \$HOME/exercises/smg2000
  - > \$HOME/exercises/lulesh2.0.3







## Open SpeedShop™



## Performance with Open | SpeedShop

## NASA Open | SpeedShop Update / Training

## Performance Analysis related to accessing Hardware Counter Information







#### Identify architectural impact on code inefficiencies





## Timing information shows where you spend your time

- Hot functions / statements / libraries
- > Hot call paths

## BUT: It doesn't show you why

- Are the computationally intensive parts efficient?
- > Are the processor architectural components working optimally?

## Answer can be very platform dependent

- > Bottlenecks may differ
- Cause of missing performance portability
- > Need to tune to architectural parameters

## Next: Investigate hardware/application interaction

- Efficient use of hardware resources or Micro-architectural tuning
- > Architectural units (on/off chip) that are stressed

## The O|SS HWC Experiments





#### Provides access to hardware counters

- Implemented on top of PAPI
- Access to PAPI and native counters
- Examples: cache misses, TLB misses, bus accesses

#### ❖ Basic model 1: Timer Based Sampling: hwcsamp

- > Samples at set sampling rate for the chosen events
- Supports multiple counters
- > Lower statistical accuracy
- Can be used to estimate good threshold for hwc/hwctime

## ❖ Basic model 2: Thresholding: hwc and hwctime

- User selects one counter
- Run until a fixed number of events have been reached
- > Take PC sample at that location
  - hwctime also records stacktrace
- > Reset number of events
- > Ideal number of events (threshold) depends on application

## **Examples of Typical Counters**





PAPI Name	Description	Threshold
PAPI_L1_DCM	L1 data cache misses	high
PAPI_L2_DCM	L2 data cache misses	high/medium
PAPI_L1_DCA	L1 data cache accesses	high
PAPI_FPU_IDL	Cycles in which FPUs are idle	high/medium
PAPI_STL_ICY	Cycles with no instruction issue	high/medium
PAPI_BR_MSP	Miss-predicted branches	medium/low
PAPI_FP_INS	Number of floating point instructions	high
PAPI_LD_INS	Number of load instructions	high
PAPI_VEC_INS	Number of vector/SIMD instructions	high/medium
PAPI_HW_INT	Number of hardware interrupts	low
PAPI_TLB_TL	Number of TLB misses	low

Note: Threshold indications are just rough guidance and depend on the application.

Note: counters platform dependent (use papi\_avail & papi\_native\_avail)

## Recommend start with HWCsamp





- 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" PAPI\_L1\_DCM,PAPI\_L1\_TCA
       50
  - Default events: PAPI\_TOT\_CYC and PAPI\_TOT\_INS
  - Default sampling\_rate: 100
  - > <PAPI\_event\_list>: Comma separated PAPI event list (Maximum of 6 events that can be combined)
  - <sampling\_rate>:Integer value sampling rate
- Use event count values to guide selection of thresholds for hwc, hwctime experiments for deeper analysis

## Selecting the Counters & Sampling Rate





## For osshwcsamp, Open | SpeedShop supports ...

- Derived and Non derived PAPI presets
  - All derived and non derived events reported by "papi\_avail"
  - Ability to sample up to six (6) counters at one time; before use test with
     papi\_event\_chooser PRESET < list of events>
  - If a counter does not appear in the output, there may be a conflict in the hardware counters
- > All native events
  - Architecture specific (incl. naming)
  - Names listed in the PAPI documentation
  - Native events reported by "papi\_native\_avail"

## Sampling rate depends on application

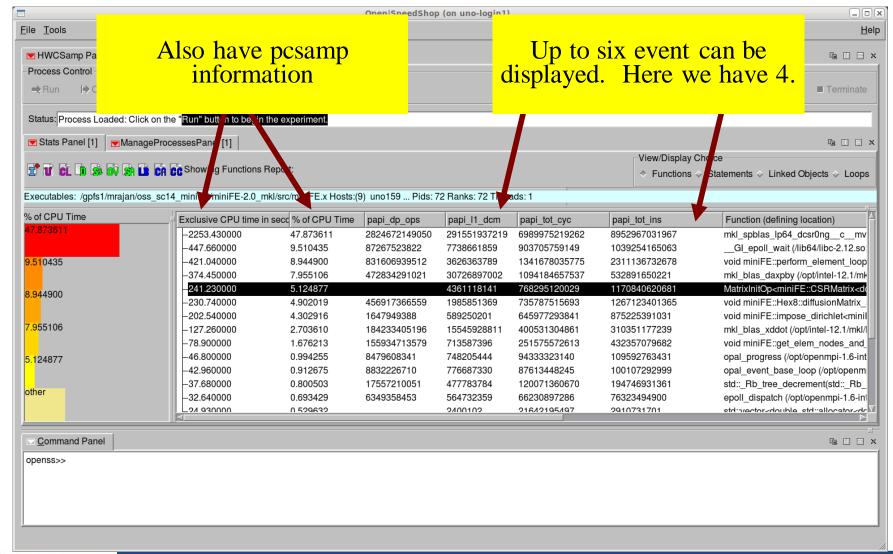
- Overhead vs. Accuracy
  - Lower sampling rate cause less samples

## hwcsamp with miniFE (see mantevo.org)





- osshwcsamp "mpiexec –n 72 miniFE.X –nx 614 –ny 614 –nz 614" PAPI\_DP\_OPS,PAPI\_L1\_DCM,PAPI\_TOT\_CYC,PAPI\_TOT\_INS
- openss –f miniFE.x-hwcsamp.openss



## Deeper Analysis with hwc and hwctime





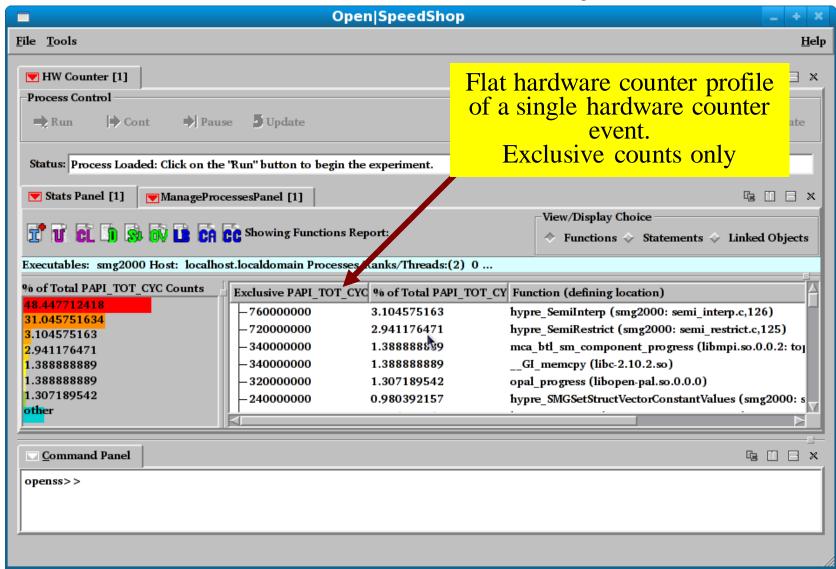
- - Sequential job example:
    - osshwc[time] "smg2000 –n 50 50 50" PAPI\_FP\_OPS 50000
  - Parallel job example:
    - osshwc[time] "mpirun –np 128 smg2000 –n 50 50 50"
- default: event (PAPI\_TOT\_CYC), threshold (10000)
- <PAPI\_event>: PAPI event name
- <PAPI threshold>: PAPI integer threshold
- NOTE: If the output is empty, try lowering the <threshold> value. There may not have been enough PAPI event occurrences to record and present

## Viewing hwc Data





### hwc default view: Counter = Total Cycles

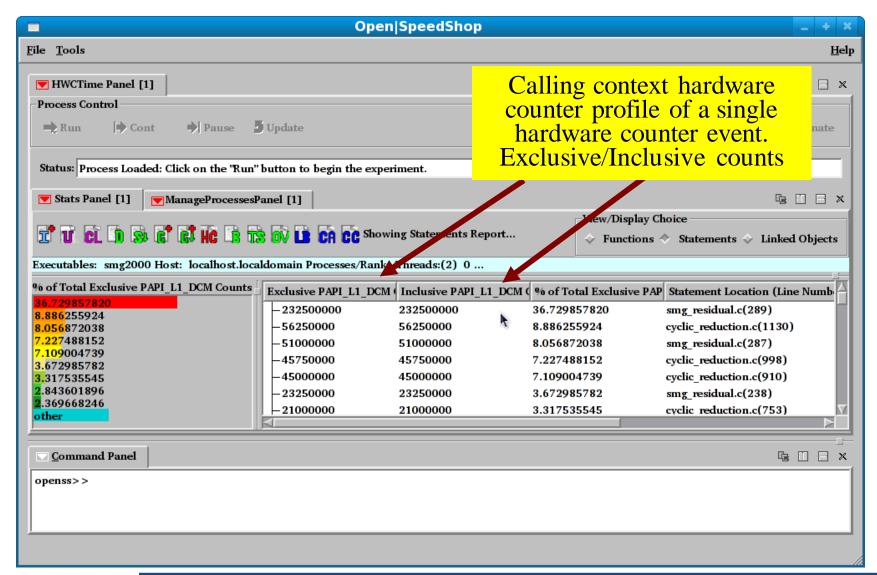


## Viewing hwctime Data





#### hwctime default view: Counter = L1 Cache Misses









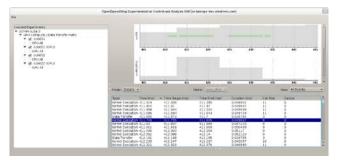
## Open | SpeedShop™ Performance with Open | SpeedShop



# NASA Open | SpeedShop Update/Training

# Performance Analysis related to application I/O activity







## I/O analysis with O|SS





## I/O Tracing (io experiment)

- > Records each event in chronological order
- Provides call path and time spent in I/O functions

## I/O Profiling (iop experiment)

- > Lighter weight I/O tracking experiment
- > Trace I/O functions but only record individual callpaths not each individual event with callpath (Like usertime)

## Extended I/O Tracing (iot experiment)

- > Records each event in chronological order
- Collects Additional Information
  - Function Parameters
  - Function Return Value
- ➤ When to use extended I/O tracing?
  - When you want to trace the exact order of events
  - When you want to see the return values or bytes read or written.
  - When you want to see the parameters of the IO call

## Running I/O Experiments





## Offline io/iop/iot experiment on sweep3d application Convenience script basic syntax:

ossio[p][t] "executable" [ default | <list of I/O func>]

- > Parameters
  - I/O Function list to sample(default is all)
  - creat, creat64, dup, dup2, lseek, lseek64, open, open64, pipe, pread, pread64, pwrite, pwrite64, read, readv, write, writev

#### **Examples:**

ossio "mpirun –np 256 sweep3d.mpi"

ossiop "mpirun –np 256 sweep3d.mpi" read,readv,write

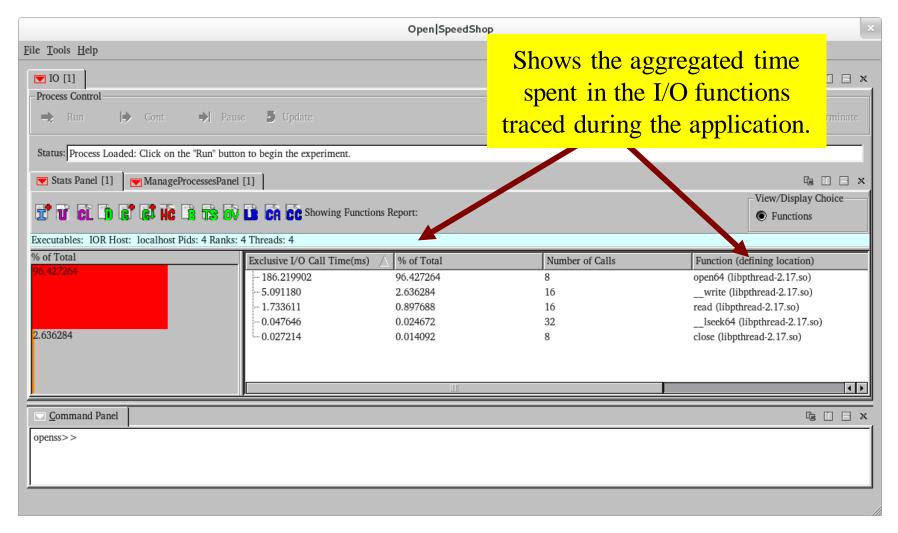
ossiot "mpirun –np 256 sweep3d.mpi" read,readv,write

## I/O output via GUI





## ❖ I/O Default View for IOR application "io" experiment

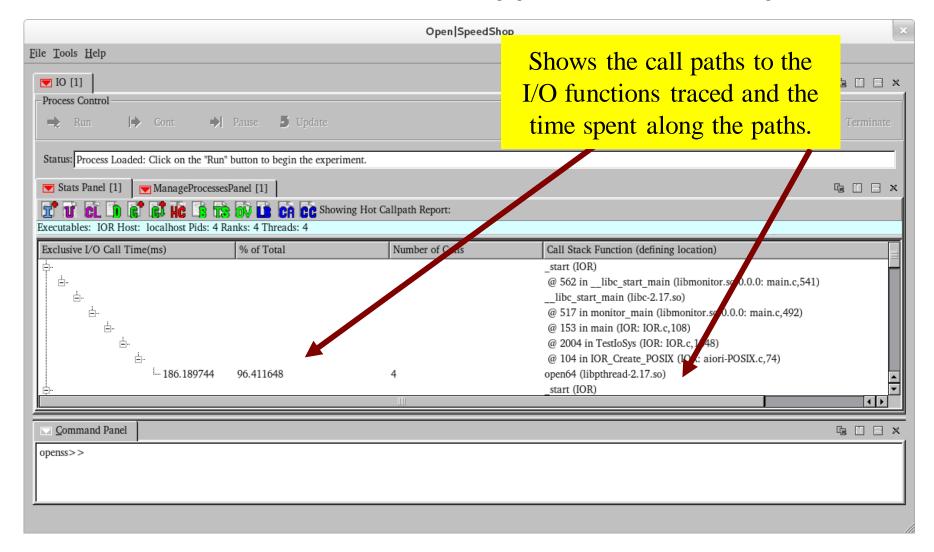


## I/O output via GUI





## ❖ I/O Call Path View for IOR application "io" experiment

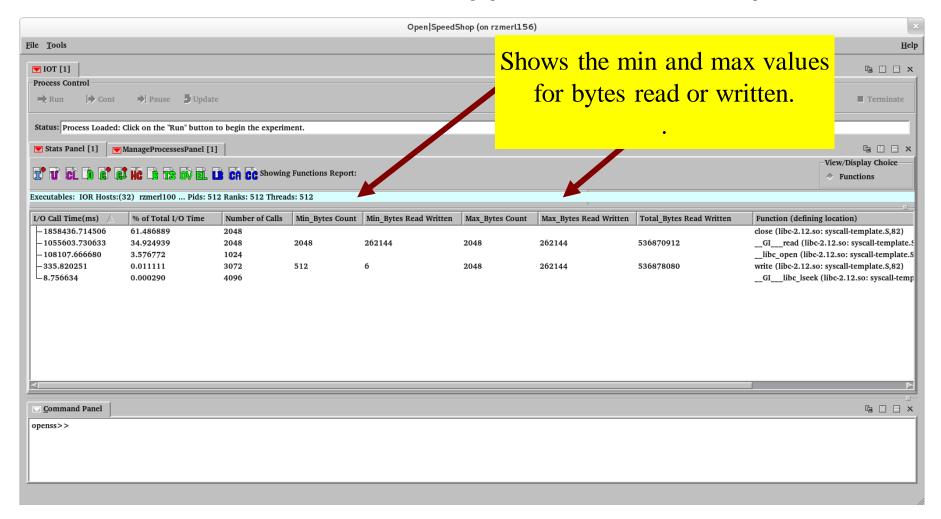


## I/O "iot" experiment output via GUI





## ❖ I/O Default View for IOR application "iot" experiment



## I/O "iot" experiment output via CLI





- Show the call paths in the application run that allocated the largest number of bytes
- Using the min\_bytes would show all the paths that allocated the minimum number of bytes.
- openss>>expview -vcalltrees,fullstack -m max\_bytes
- Max\_Bytes Call Stack Function (defining location)
- Read
- ❖ Written
  - > start (IOR)
  - > > @ 562 in \_\_libc\_start\_main (libmonitor.so.0.0.0: main.c,541)
  - > > @ 258 in \_\_libc\_start\_main (libc-2.12.so: libc-start.c,96)
  - >>> @ 517 in monitor\_main (libmonitor.so.0.0.0: main.c,492)
  - >>>> @ 153 in main (IOR: IOR.c,108)
  - >>>>> @ 2013 in TestloSys (IOR: IOR.c,1848)
  - >>>>> @ 2608 in WriteOrRead (IOR: IOR.c,2562)
  - >>>>>> @ 244 in IOR Xfer POSIX (IOR: aiori-POSIX.c,224)
  - >>>>>> @ 321 in write (iot-collector-monitor-mrnet-mpi.so: wrappers.c,239)
- 262144 >>>>>> @ 82 in write (libc-2.12.so: syscall-template.S,82)







## Open | SpeedShop™ Performance with Open | SpeedShop



# NASA Open | SpeedShop Update/Training

Parallel Performance Analysis including analysis related to application MPI and/or OpenMP activity







## How can O|SS help for parallel jobs?





## O | SS is designed to work on parallel jobs

- > Support for threading and message passing
- > Automatically tracks all ranks and threads during execution
- > Records/stores performance info per process/rank/thread

### All experiments can be used on parallel jobs

> O|SS applies the experiment collector to all ranks or threads on all nodes

#### MPI specific tracing experiments

- > Tracing of MPI function calls (individual, all, or a specific group)
- > Four forms of MPI tracing experiments

## OpenMP specific experiment (ossomptp)

- Uses OMPT API to record task time, idleness, barrier, and wait barrier per OpenMP parallel region
  - Shows load balance for time
  - expcompare time across all threads

## Analysis of Parallel Codes





#### 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
  - Note: can take a long time for large numbers of processors (current version)

## Integration with MPI





## ❖ O|SS has been tested with a variety of MPIs

➤ Including: Open MPI, MVAPICH[2], and MPICH (Intel, Cray), MPT (SGI)

## Running O | SS experiments on MPI codes

- > Just use the convenience script corresponding to the data you want to gather and put the command you use to run your application in quotes:
  - osspcsamp "mpirun –np 32 sweep3d.mpi"
  - ossio "srun –N 4 –n 16 sweep3d.mpi"
  - osshwctime "mpirun –np 128 sweep3d.mpi"
  - ossusertime "srun –N 8 –n 128 sweep3d.mpi"
  - osshwc "mpirun –np 128 sweep3d.mpi"

## MPI/OpenMP Specific Experiments





#### MPI specific experiments

- ➤ Record all MPI call invocations 100 or so that O|SS traces
- MPI functions are profiled (ossmpip)
  - Show call paths for each MPI unique call path
    - However individual call information is not recorded.
    - Less overhead than mpi, mpit.
- MPI functions are traced (ossmpi)
  - Record call times and call paths for each event
- MPI functions are traced with details (ossmpit)
  - Record call times, call paths and argument info for each event

## OpenMP specific experiment (ossomptp)

- Uses OMPT API to record task time, idleness, barrier, and wait barrier per OpenMP parallel region
  - Shows load balance for time
  - Can use CLI command: **expcompare** to compare time across all threads

## Running MPI Specific Experiments





#### Offline mpi/mpip/mpit experiment

#### **Convenience script basic syntax:**

ossmpi[t][p] "mpi executable syntax" [default | <list MPI func> | mpi category]

- Parameters
  - Default is all MPI Functions Open|SpeedShop traces
  - MPI Function list to trace (comma separated)
    - MPI Send, MPI Recv, ....
  - mpi category:
    - "all", "asynchronous\_p2p", "collective\_com", "datatypes", "environment", "graphs\_contexts\_comms", "persistent\_com", "process\_topologies", "synchronous\_p2p"

#### **Examples:**

ossmpi "srun –N 4 –n 32 smg2000 –n 50 50 50" ossmpi "mpirun –np 4000 nbody" MPI\_Send, MPI\_Recv

## Identifying Load Imbalance With O|SS





## Get overview of application

- > Run a lightweight experiment to verify performance expectations
  - pcsamp, usertime, hwc

#### Use load balance view on pcsamp, usertime, hwc

- Look for performance values outside of norm
  - Somewhat large difference for the min, max, average values
  - If the MPI libraries are showing up in the load balance for pcsamp, then do an MPI specific experiment

### Use load balance view on MPI experiment

- > Look for performance values outside of norm
  - Somewhat large difference for the min, max, average values
- > Focus on the MPI functions to find potential problems

## Use load balance view on OpenMP experiment (omptp)

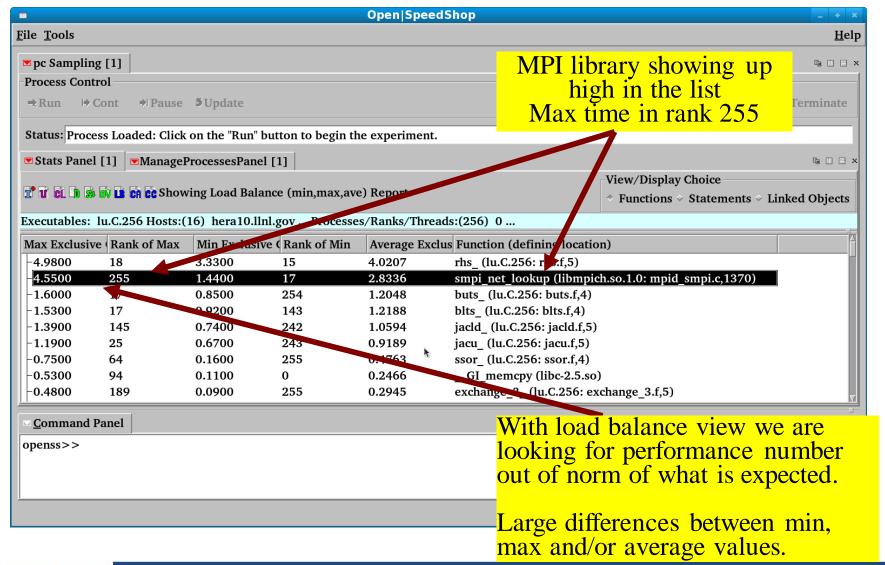
> Can also use expcompare across OpenMP threads

## Load Balance View: NPB: LU





### Load Balance View based on functions (pcsamp)

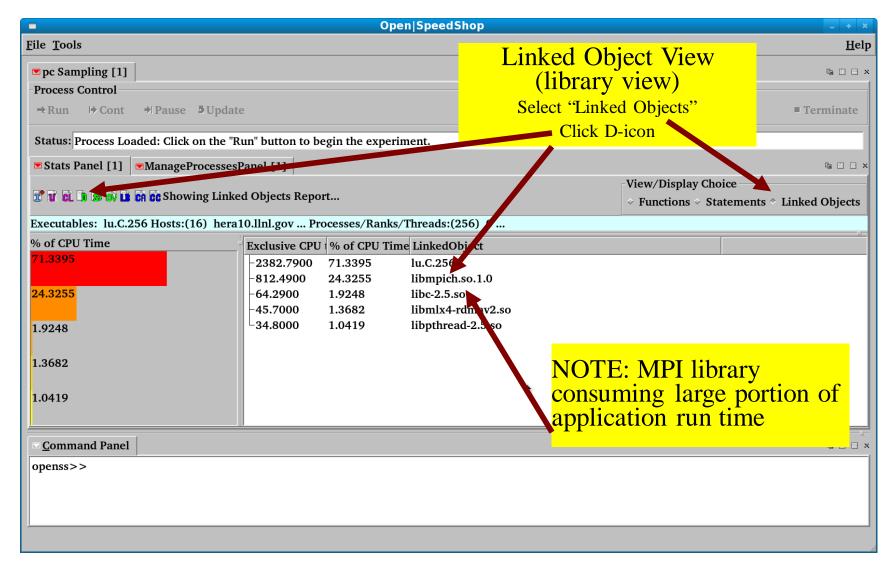


## Default Linked Object View: NPB: LU





#### Default Aggregated View based on Linked Objects (libraries)

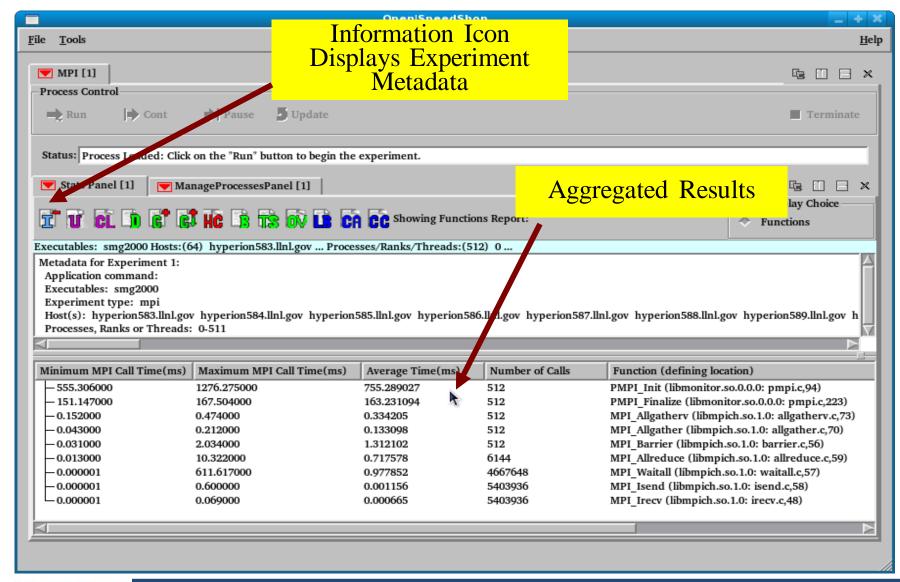


## MPI Tracing Results: Default View





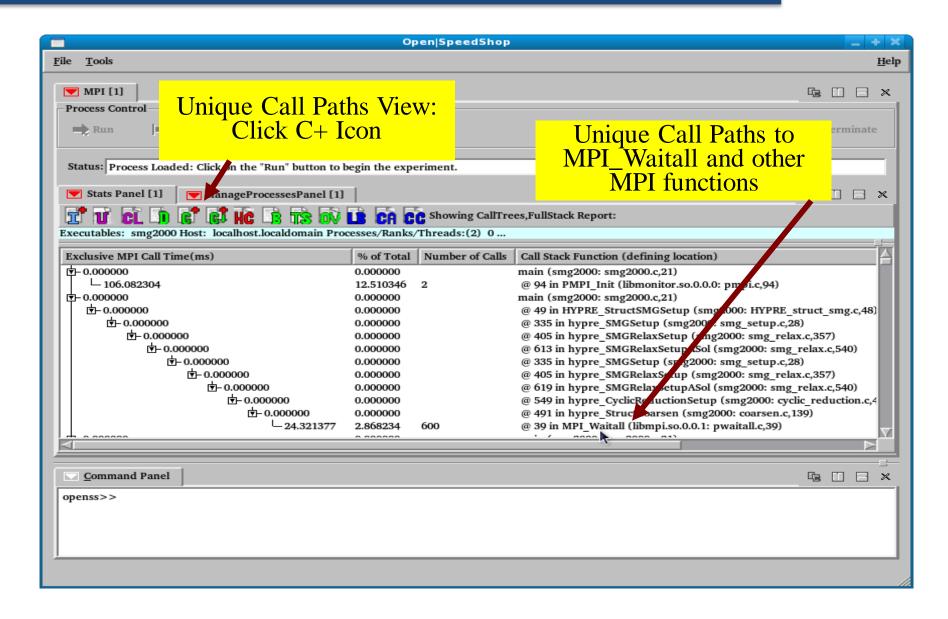
## Default Aggregated MPI Experiment View



### View Results: Show MPI Callstacks







## Using Cluster Analysis in O|SS





#### Can use with pcsamp, usertime, hwc

- Will group like performing ranks/threads into groups
- > Groups may identify outlier groups of ranks/threads
- > Can examine the performance of a member of the outlier group
- Can compare that member with member of acceptable performing group

#### Can use with mpi, mpit

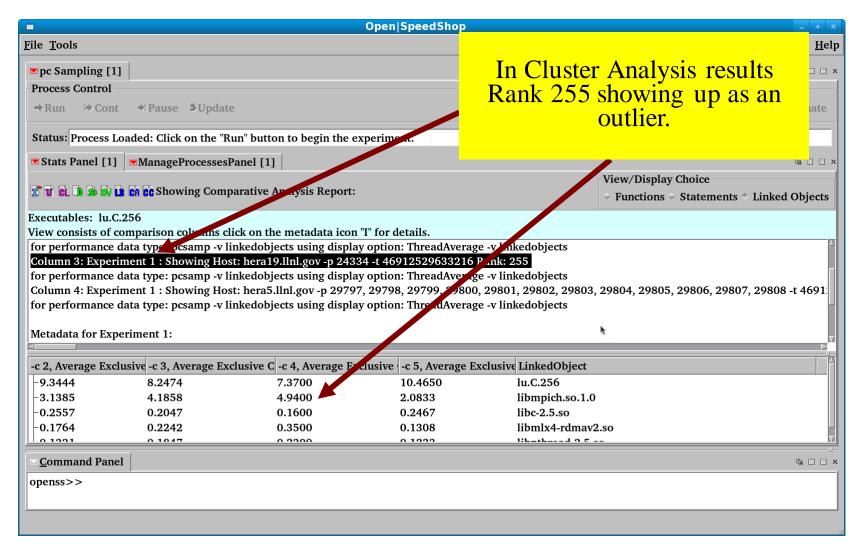
- > Same functionality as above w.r.t. cluster analysis
- But, now focuses on the performance of individual MPI\_Functions.
- Key functions are MPI\_Wait, MPI\_WaitAll
- > Can look at call paths to the key functions to analyze why they are being called to find performance issues

## Link. Obj. Cluster Analysis: NPB: LU





## Cluster Analysis View based on Linked Objects (libraries)









# Open SpeedShop™



## Performance with Open | SpeedShop

# NASA Open | SpeedShop Update / Training

## **Comparing Performance Data**







## **Comparing Performance Data**





## Key functionality for any performance analysis

- Absolute numbers often don't help
- > Need some kind of baseline / number to compare against

## Open | SpeedShop has support to line up profiles

- > Perform multiple experiments and create multiple databases
- Script to load all experiments and create multiple columns

#### Typical Example Comparisons

- Between experiments to study improvements/changes
- > Between ranks/threads to understand differences/outliers
- Before/after optimization
- Different configurations or inputs

## Comparing Performance Data in O|SS





#### Convenience Script: osscompare

- Compares Open | SpeedShop up to 8 databases to each other
- Syntax: osscompare "db1.openss,db2.openss,..." [options]
  - osscompare man page has more details
- Produces side-by-side comparison listing
- > Metric option parameter:
  - Compare based on: time, percent, a hwc counter, etc.
- Limit the number of lines by "rows=nn" option
- Specify the: viewtype=[functions|statements|linkedobjects]
  - View granularity: function, statement, or library level.
  - Function level is the default.
  - If statements option is specified:
    - Comparisons will be made by looking at the performance of each statement in all the databases that are specified.
    - Similar for libraries, if linkedobject is selected as the viewtype parameter.
- Options to write comparison output to comma separated list (csv) or text files

## Comparison Report in O|SS





#### 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 -c 4, Exclusive CPU Function (defining location)
                   time in seconds.
 time in seconds.
    3.870000000
                      3.630000000 hypre SMGResidual (smg2000: smg_residual.c,152)
    2.610000000
                      2.860000000 hypre CyclicReduction (smg2000: cyclic reduction.c,757)
                      0.15000000 opal progress (libopen-pal.so.0.0.0)
    2.030000000
    1.330000000
                      0.10000000 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
                      0.04000000 mca pml ob1 progress (libmpi.so.0.0.2:
topo unity component.c,0)
```







## Open | SpeedShop™ Performance with Open | SpeedShop



# NASA Open | SpeedShop Update/Training

## Section 2: Recently added Functionality/Experiments







### Outline





#### Section 1: Introduction to Open | SpeedShop tools

- How to use Open | SpeedShop to gather and display
- Overview of performance experiments
  - Sampling Experiments and Tracing Experiments
- How to compare performance data for different application runs

## **Section 2: New Functionality/Experiments**

- > Memory (ossmem) experiment
- OpenMP augmentation
- OMPTP (ossomptp) experiment
- > POSIX threads (osspthreads) experiment
- Lightweight experiments (ossiop, ossmpip)
- NVIDIA CUDA tracing experiment (osscuda)

#### **Section 3: Roadmap / Future Plans**

#### **Supplemental Information**

Command Line Interface (CLI) tutorial and examples







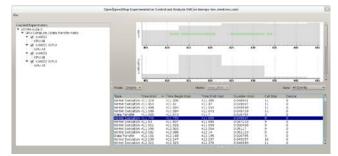
## Open | SpeedShop™ Performance with Open | SpeedShop



# NASA Open | SpeedShop Update | Training

# Performance Analysis related to application memory function activity







## O SS Memory Experiment





## Supports sequential, mpi and threaded applications.

- > No instrumentation needed in application.
- > Traces system calls via wrappers
  - malloc
  - calloc
  - realloc
  - free
  - memalign and posix\_memalign

#### Provides metrics for

- Timeline of events that set an new high-water mark.
- > List of event allocations (with calling context) to leaks.
- > Overview of all unique callpaths to traced memory calls that provides max and min allocation and count of calls on this path.

#### Example Usage

- > ossmem "./lulesh2.0"
- ossmem "mpiexec\_mpt -np 64 ./sweep3d.mpi"

#### ❖ No GUI support at this time

Support planned via the new GUI, pending funding.

## O|SS Memory Experiment CLI commands





#### expview -vunique

Show times, call counts per path, min,max bytes allocation, total allocation to all unique paths to memory calls that the mem collector saw

#### expview -vleaked

Show function view of allocations that were not released while the mem collector was active

#### expview -vtrace,leaked

Will show a timeline of any allocation calls that were not released

#### expview -vfullstack,leaked

Display a full callpath to each unique leaked allocation

#### expview -v trace, highwater

- > Is a timeline of mem calls that set a new high-water
- The last entry is the allocation call that the set the high-water for the complete run
- ➤ Investigate the last calls in the timeline and look at allocations that have the largest allocation size (size1,size2,etc) if your application is consuming lots of system ram

## O|SS Memory Experiment





## Shows the last 8 allocation events that set the high water mark

openss>>expview -vtrace,highwater

```
Start Time(d:h:m:s) Event
                                        Ptr Return Value New Call Stack Function (defining location)
                           Size
                                  Size
                                 Arg2
                      lds
                           Arg1
                                        Arg
                                                         Highwater
*** trimmed all but the last 8 events of 61 ****
2016/11/10 09:56:50.824 11877:0
                                  2080
                                                0x7760e0 19758988 >>>>> GI libc malloc (libc-
2.18.so)
2016/11/10 09:56:50.826 11877:0 1728000
                                                0x11783d0 21484908 >>>> GI libc malloc (libc-
2.18.so)
2016/11/10 09:56:50.827 11877:0 1728000
                                                0x131e1e0 23212908 >>>> GI libc malloc (libc-
2.18.so)
2016/11/10 09:56:50.827 11877:0 1728000
                                                0x14c3ff0 24940908 >>>> GI libc malloc (libc-
2.18.so)
2016/11/10 09:56:50.827 11877:0
                                   2080
                                         0
                                                 0x776a90 24942988 >>>>> GI libc malloc (libc-
2.18.so)
2016/11/10 09:56:50.919 11877:0 1728000
                                                0x1654030 25286604 >>> GI libc malloc (libc-
2.18.so)
2016/11/10 09:56:50.919 11877:0 1728000
                                                 0x17f9e40 27014604 >>>> GI libc malloc (libc-
2.18.so)
2016/11/10 09:56:50.919 11877:0
                                   2080
                                                 0xabc6a0 27016684 >>>>> GI libc malloc (libc-
                                         0
2.18.so)
```

## O|SS Memory Experiment





❖ The next slide shows the default view of all unique memory calls seen while the mem collector was active. This is an overview of the memory activity. The default display is aggregated across all processes and threads. Ability to view specific processes or threads.

#### For all memory calls the following are displayed:

- > The exclusive time and percent of exclusive time
- > The number of times this memory function was called.
- > The traced memory function name.

#### For allocation calls (e.g. malloc) the follow:

- > The max and min allocation size seen.
- > The number of times the that max or min was seen are displayed.
- > The total allocation size of all allocations.

## O|SS Memory Experiment (Unique Calls)





#### openss>>expview -vunique

Exclusive (ms)	% of Total Time	Number of Calls		Min Requested Bytes	Max Request Count	•	Total Bytes Requested	Function (defining location)
0.024847	89.028629	1546	1	192	6	4096	6316416	GIlibc_malloc (libc-2.18.so)
0.002371	8.495467	5						GIlibc_free (libc-2.18.so)
0.000369	1.322154	1	1	40	1	40	40	realloc (libc-2.18.so)
0.000322	1.153750	3	1	368	1	368	1104	calloc (libc-2.18.so)

NOTE: Number of Calls means the number of unique paths to the memory function call.

To see the paths use the CLI command: expview –vunique,fullstack

## O|SS Memory Experiment (Leaked Calls)





In this example the sequential OpenMP version of lulesh was run under ossmem.

The initial run detected 69 potential leaks of memory.

Examining the calltrees using the cli command "expview-vfullstack,leaked-mtot\_bytes" revealed that allocations from the Domain::Domain constructor where not later released in the Domain::~Domain destructor. After adding appropriate delete's in the destructor and rerunning ossmem, we observed a resolution of the leaks detected in the Domain class. The remaining leaks where minor and from system libraries.

Using the exprestore command to load in the initial database and the database from the second run, we can use the expcompare cli command to see the improvements. Below, database -x1 shows the initial run and -x2 shows the results from the run with the changes to address the leaks detected in the Domain class.

```
openss>>exprestore -f lulesh-mem-initial.openss
openss>>exprestore -f lulesh-mem-improved.openss
openss>>expcompare -vleaked -mtot_bytes -mcalls -x1 -x2
```

```
-x 2, Function (defining location)
 -x 1,
          -x 1, -x 2,
        Number Total
 Total
                         Number
          of
                           of
 Bytes
                 Bytes
Requested Calls Requested Calls
                            8 GI __libc_malloc (libc-2.17.so)
10599396
           69
                3332
                            1 realloc (libc-2.17.so)
      72
            1
                  72
```

## O|SS Memory Experiment (Highwater Calls)





# See callstacks associated with the high-water creating allocations # Note the size of the allocation, rank/thread ids, high-water mark, along with the call path: expView -vtrace, highwater, fullstack This allows the user to see the call paths to the high water allocations.

```
Size Size Call Stack Function (defining location)
  Arg Arg
        start (lulesh2.0)
       > @ 562 in __libc_start_main (libmonitor.so.0.0.0: main.c,541)
       >> libc start main (libc-2.17.so)
       >>> @ 768 in main (lulesh2.0: lulesh.cc,2690)
1728000 0 >>>> GI libc malloc(libc-2.17.so)
        start (lulesh2.0)
       > @ 562 in libc start main (libmonitor.so.0.0.0: main.c,541)
       >> libc start main (libc-2.17.so)
       >>> @ 176 in main (lulesh2.0: lulesh.cc,2690)
1728000 0 >>>> GI libc malloc(libc-2.17.so)
        start (lulesh2.0)
       > @ 562 in libc start main (libmonitor.so.0.0.0: main.c,541)
       >> libc start main (libc-2.17.so)
       >>> @ 176 in main (lulesh2.0: lulesh.cc,2690)
1728000 0 >>>> GI libc malloc(libc-2.17.so)
        start (lulesh2.0)
       > @ 562 in libc start main (libmonitor.so.0.0.0: main.c,541)
       >> libc start main (libc-2.17.so)
       >>> @ 176 in main (lulesh2.0: lulesh.cc,2690)
1728000 0 >>>> GI libc malloc(libc-2.17.so)
```

## **Summary and Conclusions**





## Benefits of Memory Heap Analysis

- > Detect leaks
- Inefficient use of system memory
- > Find potential OOM, paging, swapping conditions
- > Determine memory footprint over lifetime of application run

## Observations of Memory Analysis Tools

- > Less concerned with the time spent in memory calls
- Emphasis is placed on the relationship of allocation calls to free calls.
- > Can slow down and impact application while running

## **Demonstration: Memory Analysis**





## If timing permits:

- Memory experiment related application exercise
  - More information provided at the tutorial
- Memory exercises can be found in these directories:
  - > \$HOME/exercises/matmul
  - > \$HOME/exercises/lulesh2.0.3
  - > \$HOME/exercises/lulesh2.0.3-fixed

**\*** Look for the README file for instructions.







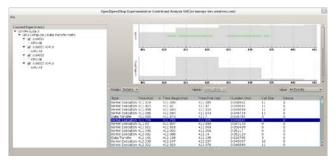
## Open | SpeedShop™ Performance with Open | SpeedShop



# NASA Open | SpeedShop Update | Training

# Performance Analysis related to application OpenMP activity











- ❖ O|SS augments the sampling experiments
  - Applies the OMPT API callbacks for:
    - ompt\_event\_thread\_idle
    - ompt\_event\_thread\_barrier
    - ompt\_event\_thread\_wait\_barrier
  - to samples taken in the OpenMP library that otherwise would be shown as
    - \_\_kmp\_barrier
    - \_\_kmp\_wait\_sleep, etc.
  - > in the Intel libiomp5 library
- The user can see the sample time per thread for idle, barrier, and wait\_barrier.
- ❖ The user can also use the loadbalance metric to see the min, max, avg of these idle, barrier, and wait barrier events
- Use the expcompare command across all threads to see individual metrics in comparison to each other





#### With respect to the barrier symbols

- > Samples taken when a thread is waiting at a barrier are inclusive to total barrier time.
- I.e. adding barrier and wait\_barrier metrics is equal to the total barrier time.
- Essentially these metrics as used in the O|SS sampling experiments to:
  - ➤ Inform the user the time a thread is idle and the time spent at a barrier (including waiting at a barrier).
- The usertime experiment can give some context to where specific idle and barrier times are.





Using the usertime experiment on an OpenMP application can help to pinpoint where in the source the wait barrier time is coming from. For example:

```
openss>>expview
Exclusive Inclusive
                     % of Function (defining location)
CPU time CPU time
                      Total
                     Exclusive
   in
             in
seconds. seconds.
                     CPU Time
23.200000 23.200000 38.648263 OMPT THREAD IDLE (usertime-collector-monitor-mrnet.so: collector.c,122)
13.142857 13.142857 21.894336 MAIN .omp fn.2 (stress omp: stress omp.f,179)
12.885714 12.885714 21.465969 MAIN .omp fn.5 (stress omp: stress omp.f,227)
 4.742857 4.742857 7.901000 OMPT THREAD WAIT BARRIER (usertime-collector-monitor-mrnet.so: collector.c,150)
 2.000000 11.771428 3.331747 MAIN_ (stress_omp:stress_omp.f,1)
 1.257143 1.257143 2.094241 kernel cosf (libm-2.12.so: k cosf.c,45)
 1.085714 1.085714 1.808663 ieee754 rem pio2f(libm-2.12.so:e rem pio2f.c,108)
```





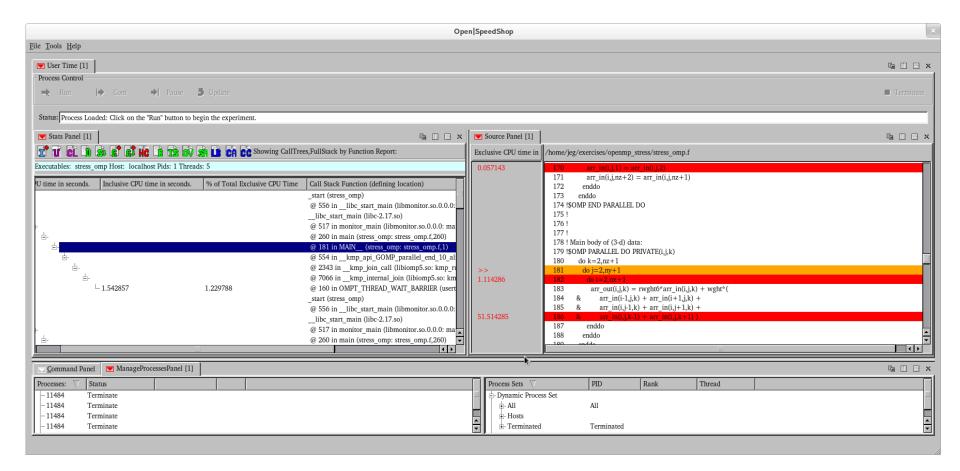
- Here we see the call path that points to the source lines that result in the thread waiting in the barrier.
- openss>>expview-vcalltrees,fullstack-f OMPT\_THREAD\_WAIT\_BARRIER usertime1

```
Exclusive Inclusive % of Call Stack Function (defining location)
                                                                                 Total
  CPU time CPU time
             in
                                   in Exclusive
  seconds. Seconds. CPU Time
                                                             start (stress omp)
                                                             > @ 556 in __libc_start_main (libmonitor.so.0.0.0: main.c,541)
                                                             >> libc start main (libc-2.12.so)
                                                             >>> @ 517 in monitor_main (libmonitor.so.0.0.0: main.c,492)
                                                             >>>main (stress omp)
                                                             >>>> @ 227 in MAIN (stress omp: stress omp.f,1)
                                                             >>>> @ 557 in __kmp_api_GOMP_parallel_end_10_alias (libiomp5.so: kmp_gsupport.c,490)
                                                             >>>>> @ 2395 in kmp join call (libiomp5.so: kmp runtime.c,2325)
                                                             >>>>> @ 7114 in __kmp_internal_join (libiomp5.so: kmp_runtime.c,7093)
                                                             >>>>> @ 1458 in __kmp_join_barrier(int) (libiomp5.so: kmp_barrier.cpp,1371)
  1.742857 1.742857 2.903379 >>>>>> @ 150 in OMPT THREAD WAIT BARRIER (MATTER CORRECTION CONTINUE CONTIN
```





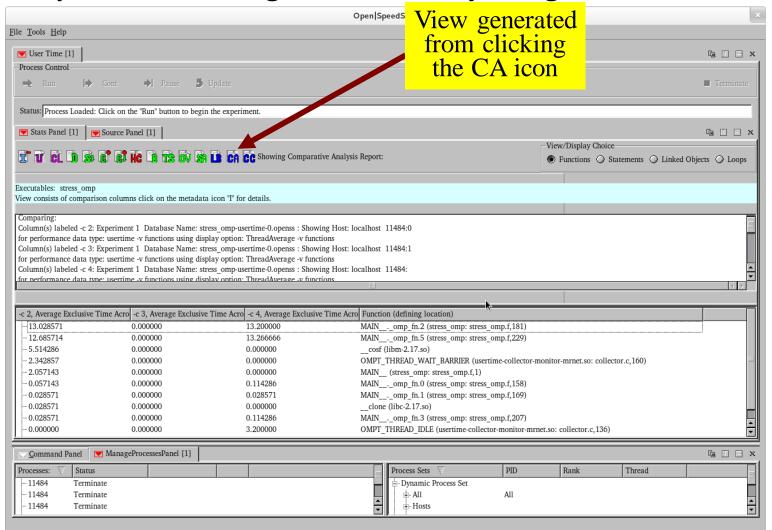
Here, in the GUI, we see the call path that points to the source lines that result in the thread waiting in the barrier.







Here, in the GUI, we see any outliers using the "CA" icon, which corresponds to running a cluster analysis algorithm.







- An interesting view for OpenMP in any of the augmented experiments would be (for 4 openmp threads):
- openss>>expcompare -t0:4 -mtime -v summary

```
-t 0, -t 2, -t 3, -t 4, Function (defining location)

Exclusive Exclusive Exclusive Exclusive

CPU time CPU time CPU time CPU time in in in in seconds. seconds. seconds. seconds. seconds. seconds. seconds. 12.771428 8.514286 4.971428 10.257143 compute._omp_fn.1 (matmult: matmult.c,68)

0.028571 0.057143 0.085714 0.114286 compute_triangular._omp_fn.2 (matmult: matmult.c,95)

0.000000 0.571429 0.142857 1.200000 OMPT_THREAD_IDLE (usertime-collector-monitor-mrnet.so: collector.c,124)

0.000000 0.400000 0.485714 0.800000 compute_interchange._omp_fn.3 (matmult: matmult.c,118)

12.800000 9.542857 5.685714 12.371428 Report Summary
```

- According to the summary, t0 the master thread had the most sample time. It should.
- Thread t4 also was used almost to the same amount as t0 but did have 1.2 seconds of idle.
- The above states that parallel region compute is not balanced in terms or time. particularly t3.
- If this was an mpi hybrid, assume these times as across all ranks. So if each rank uses 4 threads, the number of ranks\*4 is the number of total cores used.





#### This display shows the total times for barrier, idle, and barrier wait

```
openss>>expcompare -mtime -t0:4 -f OMPT*
```

```
-t 0, -t 2, -t 3, -t 4, Function (defining location)

Exclusive Exclusive Exclusive Exclusive

CPU time CPU time CPU time CPU time in in in in seconds. seconds. seconds. seconds. seconds. seconds. seconds.

2.514286 0.342857 0.000000 0.228571 OMPT_THREAD_WAIT_BARRIER (usertime-collector-monitor-mrnet.so: collector.c,148)

0.000000 5.285714 2.028571 3.828571 OMPT_THREAD_IDLE (usertime-collector-monitor-mrnet.so: collector.c,124)

0.000000 0.000000 0.000000 0.0028571 OMPT_THREAD_BARRIER (usertime-collector-monitor-mrnet.so: collector.c,136)
```





Here is the CLI view from expcompare of calltrees all due to the calltree difference that was shown above between the master and worker thread calltrees.

openss>>expcompare -mtime -t0:4 -f OMPT\* -v calltrees

```
-t 2, -t 3, -t 4, Call Stack Function (defining location)
Exclusive Exclusive Exclusive
CPU time CPU time CPU time
   in
         in
seconds, seconds, seconds,
                    start (LULESH OMP)
                    > libc_start_main (libmonitor.so.0.0.0: main.c,541)
                    >> libc start main (libc-2.18.so)
                    >>>monitor_main (libmonitor.so.0.0.0: main.c,492)
                    >>>main (LULESH OMP: LULESH OMP.cpp,2961)
                    >>>>LagrangeLeapFrog() (LULESH_OMP: LULESH_OMP.cpp,2946)
                    >>>>LagrangeNodal() (LULESH_OMP: LULESH_OMP.cpp,1687)
                    >>>>CalcForceForNodes() (LULESH OMP: LULESH OMP.cpp,1596)
                    >>>>>CalcVolumeForceForElems() (LULESH OMP: LULESH OMP.cpp,1562)
                    >>>>>CalcHourglassControlForElems(double*, double) (LULESH_OMP: LULESH_OMP.cpp,1505)
                    >>>>>CalcFBHourglassForceForElems(double*, double*, double*, double*, double*, double*, double*, double*, double
(LULESH_OMP: LULESH_OMP.cpp,1238)
                    >>>>>> kmp_api_GOMP_parallel_end_10_alias (libiomp5.so: kmp_gsupport.c,492)
                    >>>>>> kmp join call (libiomp5.so: kmp runtime.c,2273)
                    >>>>>> kmp_internal_join (libiomp5.so: kmp_runtime.c,7045)
                    >>>>>> kmp join barrier(int) (libiomp5.so: kmp barrier.cpp,1371)
0.485714
                        >>>>>>OMPT_THREAD_WAIT_BARRIER (usertime-collector-monitor-mrnet.so: collector.c,148)
                    >>>>LagrangeElements() (LULESH OMP: LULESH OMP.cpp,2786)
                    >>>>>CalcQForElems() (LULESH_OMP: LULESH_OMP.cpp,2353)
                    >>>>>CalcMonotonicQForElems() (LULESH OMP: LULESH OMP.cpp,2324)
                    >>>>>CalcMonotonicQRegionForElems(double, double, double, double, double, int) (LULESH OMP: LULESH OMP.cpp,2192)
                    >>>>>> kmp api GOMP parallel end 10 alias (libiomp5.so: kmp gsupport.c,492)
                    >>>>>> kmp_join_call (libiomp5.so: kmp_runtime.c,2273)
                    >>>>>> kmp internal join (libiomp5.so: kmp runtime.c,7045)
                    >>>>>> kmp_join_barrier(int) (libiomp5.so: kmp_barrier.cpp,1371)
0.257143
                        >>>>>>OMPT THREAD WAIT BARRIER (usertime-collector-monitor-mrnet.so: collector.c,148)
                    >>>>CalcLagrangeElements(double) (LULESH_OMP: LULESH_OMP.cpp,2005)
                    >>>>>CalcKinematicsForElems(int, double) (LULESH OMP: LULESH OMP.cpp,1929)
                    >>>>> kmp_api_GOMP_parallel_end_10_alias (libiomp5.so: kmp_gsupport.c,492)
                    >>>>>> kmp join call (libiomp5.so: kmp runtime.c,2273)
                    >>>>>> kmp internal join (libiomp5.so: kmp runtime.c,7045)
                    >>>>>> kmp_join_barrier(int) (libiomp5.so: kmp_barrier.cpp,1371)
                        >>>>>>OMPT THREAD WAIT BARRIER (usertime-collector-monitor-mrnet.so: collector.c,148)
0.257143
```

## Using OMPTP experiment in O|SS





The following three CLI examples show the most important ways to view OMPTP experiment data.

No GUI support at this time. Support planned via the new GUI, pending funding.

Default view shows the timing of the parallel regions, idle, barrier, and wait barrier as an aggregate across all threads

```
openss>>expview
                  % of Function (defining location)
Exclusive Inclusive
times in times in
                   Total
seconds, seconds, Exclusive
           CPU Time
44.638794 45.255843 93.499987 compute. omp fn.1 (matmult: matmult.c,68)
                     3.654726 compute_interchange._omp_fn.3 (matmult: matmult.c,118)
 1.744841 1.775104
 0.701720 0.701726 1.469817 compute triangular. omp fn.2 (matmult: matmult.c,95)
 0.652438  0.652438  1.366591 IDLE (omptp-collector-monitor-mrnet.so: collector.c,573)
                     0.008810 initialize. omp fn.0 (matmult: matmult.c,32)
 0.004206 0.009359
 0.000032 0.000032
                     0.000068 BARRIER (omptp-collector-monitor-mrnet.so: collector.c,587)
 0.000000 0.000000
                     0.000001 WAIT BARRIER (omptp-collector-monitor-mrnet.so: collector.c,602)
```

openss -cli -f ./matmult-omptp-0.openss

## Using OMPTP experiment in O|SS





#### This example shows the comparison of exclusive time across all threads for the parallel regions, idle, barrier, and wait barrier

openss>>expcompare -mtime -t0:4

```
-t 2, -t 3, -t 4, Function (defining location)
Exclusive Exclusive Exclusive
times in times in times in
seconds. seconds. seconds.
11.313892 11.081346 11.313889 10.929668 compute._omp_fn.1 (matmult: matmult.c,68)
                               0.440940 compute_interchange._omp_fn.3 (matmult: matmult.c,118)
0.443713  0.430553  0.429635
                               0.069975 compute triangular. omp fn.2 (matmult: matmult.c,95)
0.253632
          0.213238
                    0.164875
                               0.000964 initialize._omp_fn.0 (matmult: matmult.c,32)
 0.001047
          0.001100
                     0.001095
 0.000008 0.000008
                     0.000006
                               0.000010 BARRIER (omptp-collector-monitor-mrnet.so: collector.c,587)
0.000000
          0.000000
                     0.000000
                               0.000000 WAIT BARRIER (omptp-collector-monitor-mrnet.so:collector.c,602)
                                0.388890 IDLE (omptp-collector-monitor-mrnet.so: collector.c,573)
 0.000000
          0.247592
                     0.015956
```

## Using OMPTP experiment in O|SS





## This example shows the load balance of time across all threads for the parallel regions, idle, barrier, and wait barrier

openss>>expview-mloadbalance

Max O <sub>l</sub>	penl	MP Min	Open	MP Average Function (defining location)					
<b>Exclusive Th</b>	read	dId Exclusive	Threa	adId Exclusive					
Time Across of Max Time Across of Min Time Across									
OpenMP		OpenMP							
ThreadIds(s)		ThreadIds(s)		ThreadIds(s)					
11.313892	0	10.929668	4	11.159699 computeomp_fn.1 (matmult: matmult.c,68)					
0.443713	0	0.429635	3	0.436210 compute_interchangeomp_fn.3 (matmult: matmult.c,118)					
0.388890	4	0.015956	3	0.217479 IDLE (omptp-collector-monitor-mrnet.so:					
collector.c,573)									
0.253632	0	0.069975	4	0.175430 compute_triangularomp_fn.2 (matmult: matmult.c,95)					
0.001100	2	0.000964	4	0.001052 initializeomp_fn.0 (matmult: matmult.c,32)					
0.000010	4	0.000006	3	0.000008 BARRIER (omptp-collector-monitor-mrnet.so:					
collector.c,587	7)								
0.000000	0	0.000000	0	0.000000 WAIT_BARRIER (omptp-collector-monitor-mrnet.so:					
collector.c.602	2)								

## Demonstration: OpenMP specific profiling





## If timing permits:

- OpenMP specific experiment application exercise
- OpenMP profiling exercises can be found in these directories:
  - > \$HOME/exercises/matmul
  - \$HOME/exercises/hybrid\_lulesh2.0.3
  - > \$HOME/exercises/lulesh2.0.3







## Open | SpeedShop™ Performance with Open | SpeedShop



# NASA Open | SpeedShop Update | Training

Performance Analysis related to application POSIX thread activity







## OSS/CBTF pthreads experiment





## pthreads experiment was created using the CBTF infrastructure

- Gives opportunity to filter the POSIX thread performance information to reduce and mine the important/worthwhile information while the data is transferring to the client tool
  - > Discussion Topic: What is that worthwhile information?
  - > Ideas:
    - Report statistics about pthread wait
    - Report OMP blocking times
    - Attribute information to proper threads
    - Thread numbering improvements
      - Use a shorter alias number for the long POSIX pthread numbers
    - Report synchronization overhead mapped to proper thread
- Slides that follow show what the tool provides currently





# OSS/CBTF pthreads experiment information and example Convenience script basic syntax:

osspthreads "executable" [ default | <list of POSIX func>]

- > Parameters
  - POSIX thread function list to sample(default is all)
  - pthread\_create, pthread\_mutex\_init, pthread\_mutex\_destroy, pthread\_mutex\_lock, pthread\_mutex\_trylock, pthread\_mutex\_unlock, pthread\_cond\_init, pthread\_cond\_destroy, pthread\_cond\_signal, pthread\_cond\_broadcast, pthread\_cond\_wait, pthread\_cond\_timedwait

#### **Examples:**

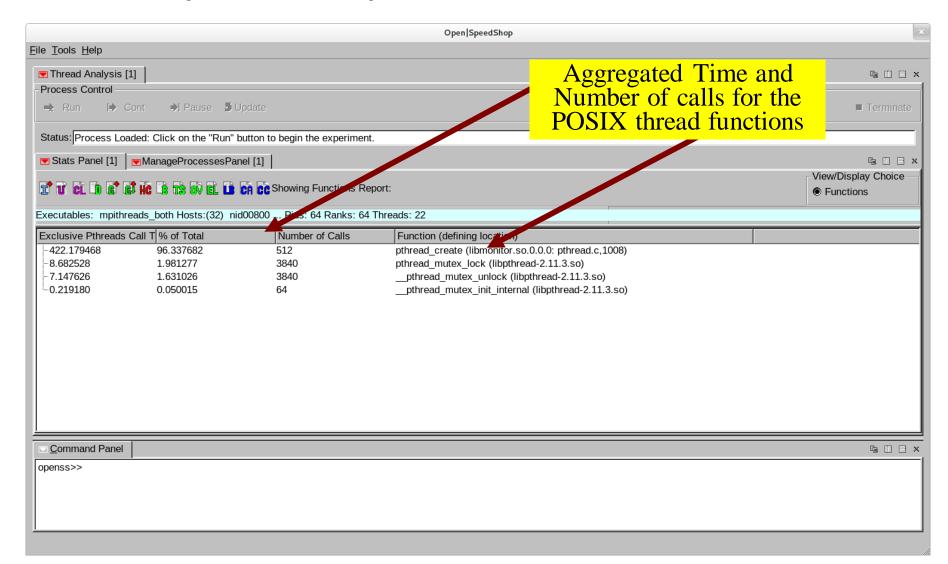
osspthreads "aprun -n 64 -d 8 ./mpithreads\_both"

osspthreads "mpirun –np 256 sweep3d.mpi" pthread\_mutex\_lock





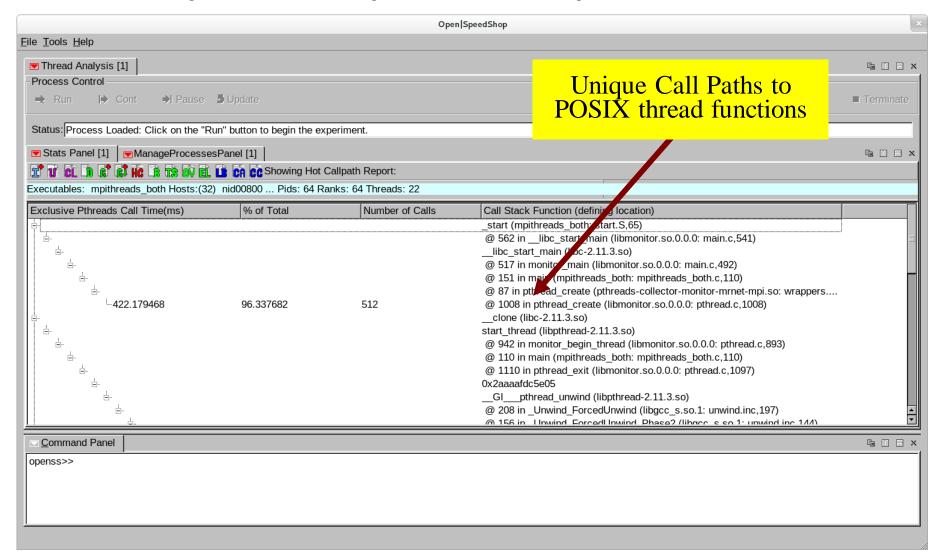
#### OSS/CBTF pthreads experiment default GUI view







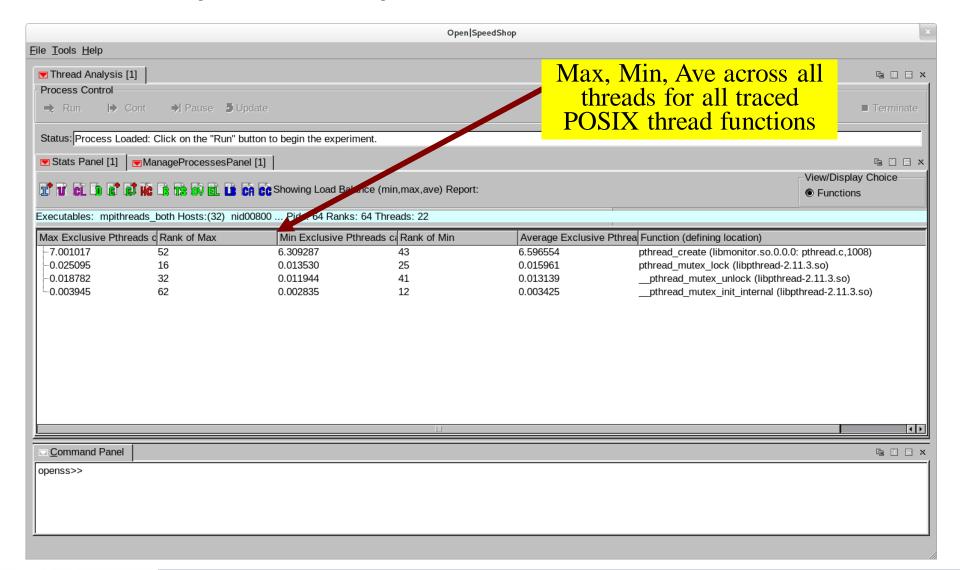
#### OSS/CBTF pthreads experiment callpath GUI view







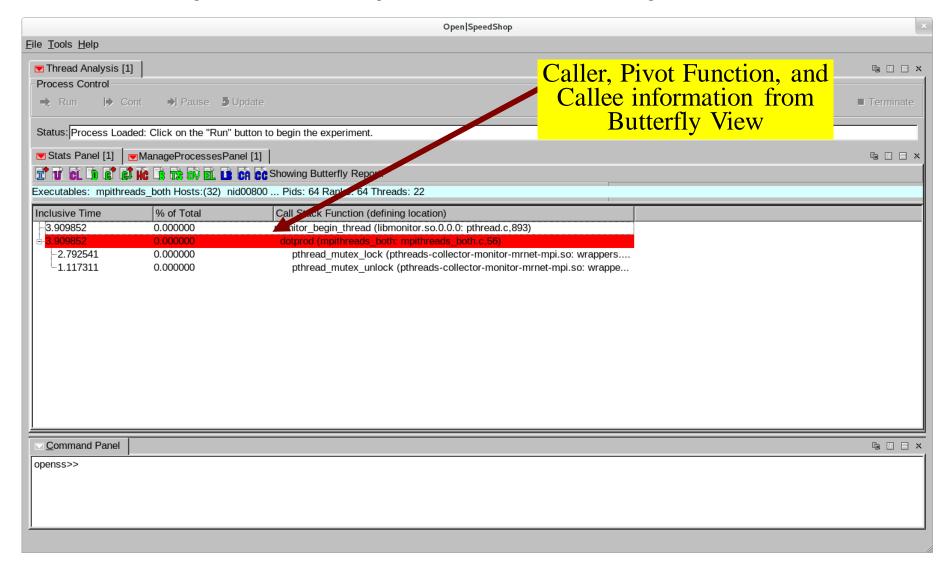
#### OSS/CBTF pthreads experiment loadbalance GUI view







#### OSS/CBTF pthreads experiment butterfly GUI view









## Open | SpeedShop™



## Performance with Open | SpeedShop

# NASA Open | SpeedShop Update | Training

## Lightweight I/O and MPI







## OSS/CBTF iop and mpip experiment





#### **Lightweight Experiments**

- ❖ iop Gather I/O information like the io experiment, but do not save the information about each individual I/O call.
- mpip Gather MPI information like the io experiment, but do not save the information about each individual MPI call.
- Experiments still give a good overview of I/O and MPI, but reduce the sizes of the Open | SpeedShop database created.

#### Size comparison:

- 668K smg2000-mpip-0.openss
  - ossmpip "mpirun -np 4 ./smg2000 -n 10 10 10"
- > 5.0M smg2000-mpi-0.openss
  - ossmpi "mpirun -np 4 ./smg2000 -n 10 10 10"
- > 12M smg2000-mpit-0.openss
  - ossmpit "mpirun -np 4 ./smg2000 -n 10 10 10"
- > 60K smg2000-mpit-1.openss
  - Gathered data for only the MPI collective mpi category.
  - ossmpit "mpirun -np 4 ./smg2000 -n 10 10 10" collective com







## Open | SpeedShop™ Performance with Open | SpeedShop



# NASA Open | SpeedShop Update/Training

## **NVIDIA CUDA Performance Analysis**







## OSS/CBTF cuda experiment





#### What performance info does O | SS provide?

- For GPGPU O|SS reports information to help understand:
  - > Time spent in the GPU device
  - Cost and size of data transferred to/from the GPU
  - Balance of CPU versus GPU utilization
  - Transfer of data between the host and device memory versus the execution of computational kernels
  - Performance of the internal computational kernel code running on the GPU device
- ❖ O|SS is able to monitor CUDA scientific libraries because it operates on application binaries.
- Support for CUDA based applications is provided by tracing actual CUDA events
- **❖** OpenACC support is conditional on the CUDA RT.

## Open | SpeedShop: osscuda experiment





#### Usage:

osscuda "executable" [extra\_args]

Where "executable" is defined as the command that you normally use to execute your program but placed in quotes.

Example: osscuda "mpiexec\_mpt -np 8 ./Triad" [extra\_args]

The optional "extra\_args" are defined as follows:

The following arguments control the periodic sampling of both CPU and GPU hardware performance counters performed by the cuda collector:

"all" - Periodically sample all instructions.

"branches" - Periodically sample branch instructions.

"integer" - Periodically sample integer instructions.

"single" - Periodically sample single-precision float instructions.

"double" - Periodically sample double-precision float instructions.

"memory" - Periodically sample load/store instructions.

"low" - Periodically sample the requested instructions every 100 ms.

"default" - Periodically sample the requested instructions every 10 ms.

"high" - Periodically sample the requested instructions every 1 ms.

#### CUDA GUI View: Default CUDA view





Note: The left pane shows the executable and the nodes it ran on. In future, will effect views. Internal GPU activity is shown in ccn0001 (GPU All) graphic (shaded area)

Red boxes indicate data transfers, Green boxes indication GPU kernel executions

Source panel displays source for metrics clicked on in the Metric pane.

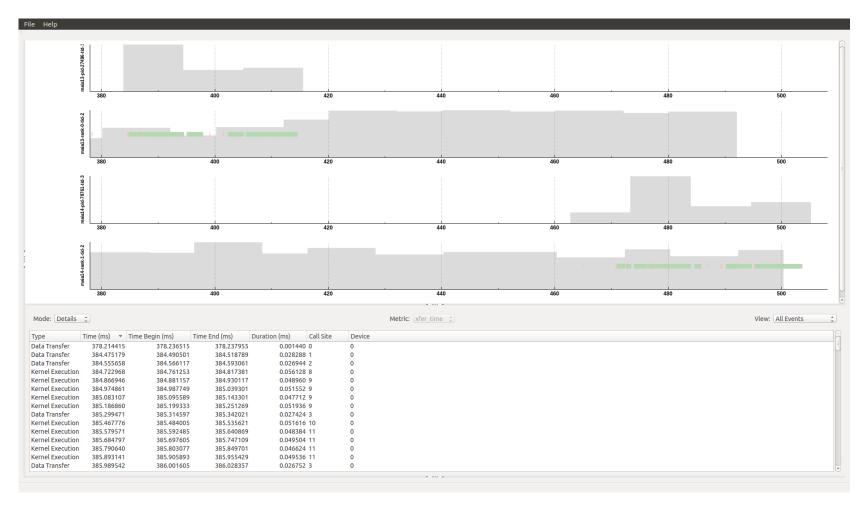


#### CUDA GUI View: All Events Trace





Note: This is the "All Events" Details View which shows the chronological list of CUDA kernel executions and data transfers. Here the Experiment Panel (the left side panel) has been completely collapsed to maximize the width of the right-side panels.



#### Outline





#### Section 1: Introduction to Open | SpeedShop tools

- How to use Open | SpeedShop to gather and display
- Overview of performance experiments
  - Sampling Experiments and Tracing Experiments
- > How to compare performance data for different application runs

### **Section 2: New Functionality/Experiments**

- Memory (ossmem) experiment
- OpenMP augmentation
- > OMPTP (ossomptp) experiment
- > POSIX threads (osspthreads) experiment
- Lightweight experiments (ossiop, ossmpip)

### **Section 3: Roadmap / Future Plans**

#### **Supplemental Information**

> Command Line Interface (CLI) tutorial and examples







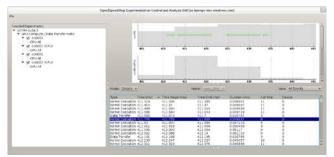
# Open | SpeedShop™ Performance with Open | SpeedShop



# NASA Open | SpeedShop Update | Training

# Section 3 Road Map / Future Work







April 27, 2017

# What are the recent changes to O|SS





### Component Based Tool Framework (CBTF)

- New version of O|SS uses tree based network (MRNet)
  - Transfer data over the network, does not write files like the offline version
  - Allows the possibility of data reduction (in parallel) as the data is streamed up the tree
- > Six new experiments implemented in this version
  - Lightweight I/O profiling (iop)
  - Lightweight MPI profiling (mpip)
  - Threading experiments (pthreads)
  - Memory usage analysis (mem)
  - GPU/Accelerator support (cuda)
  - OpenMP specific support (omptp)

# What are the recent changes to O|SS





#### New features and improvements in O | SS

- > OpenMP idle/wait time augmentation to sampling experiments
- Spack build support for clusters not Cray yet
- Conversion to cmake builds for O|SS, CBTF from GNU auto tools.
- > Support for offline like capability in the O|SS CBTF version
  - Use osspcsamp --offline "how you run your application normally"
  - Same for other experiments: ossusertime, osshwc, etc..
- ➤ Fix in Qt3 GUI for better support of function related views when function name is STL or C++ namespace based.
- > Major improvements to NVIDIA CUDA GPU experiment
  - Initial new GUI creation
  - Improved performance data collection
  - Improved command line interface (CLI) views
- > ARM, Power8 support
- Tracing of MPI asynchronous non-blocking functions in the MPI experiments.

# Open | SpeedShop and CBTF





#### New functionality being worked on now or planned

- Creation of an Overview experiment
  - Give users an overview of the performance of their application
  - Include information in a lightweight manner
  - Include MPI, I/O, hardware counters, PC sampling, other
  - > May not create a database? Still in the planning stages
  - > Task for Tri-labs listed in development contract
- Continue improving Intel MIC (KNL) support
- Filtering (data reduction, analysis) in the MRNet communication nodes
  - Faster views as data is mined in parallel
- Investigate performance analysis by phases and iteration of the phase, perhaps using LLNL caliper project.
- Spack based OpenSpeedShop builds for Cray platform
- In discussion: replacement/upgrade for mpiotf experiment to write OTF-2 instead of OTF. OTF == Open Trace Format

# NASA Open | SpeedShop Availability





#### **Pleaides platform:**

- module use /home4/jgalarow/privatemodules
- Module names:
  - module load openspeedshop (defaults to mpt)
  - > module load openspeedshop.mpt
  - > module load openspeedshop.intelmpi
  - > module load openspeedshop.mvapich2
  - > module load openspeedshop.openmpi

#### **KNL** cluster platform:

- module use /u/jgalarow/privatemodules
- Module names:
  - module load openspeedshop (defaults to mpt)
  - > module load openspeedshop.mpt
  - > module load openspeedshop.intelmpi

For mpi\* experiments use the module file that corresponds to the MPI implementation your application was built with.

# **Availability**





- Current version: 2.3.1 has been released
- Open|SpeedShop Website
  - http://www.openspeedshop.org/
- Open | SpeedShop help and bug reporting
  - > <u>Direct email: oss-contact@openspeedshop.org</u>
  - Forum/Group: oss-questions@openspeedshop.org

#### ❖ Feedback

- > Bug tracking available from website
- > Feel free to contact presenters directly
- > Support contracts and onsite training available

# Open | SpeedShop Documentation





- Build and Installation Instructions
  - http://www.openspeedshop.org/documentation
    - Look for: Open | SpeedShop Version 2.3 Build/Install Guide
- Open | SpeedShop User Guide Documentation
  - http://www.openspeedshop.org/documentation
    - Look for Open | SpeedShop Version 2.3 Users Guide
- Man pages: OpenSpeedShop, osspcsamp, ossmpi,
- Quick start guide downloadable from web site
  - http://www.openspeedshop.org
  - > Click on "Download Quick Start Guide" button

#### Outline





#### Section 1: Introduction to Open | SpeedShop tools

- How to use Open | SpeedShop to gather and display
- Overview of performance experiments
  - Sampling Experiments and Tracing Experiments
- > How to compare performance data for different application runs

### **Section 2: New Functionality/Experiments**

- Memory (ossmem) experiment
- OpenMP augmentation
- > OMPTP (ossomptp) experiment
- > POSIX threads (osspthreads) experiment
- Lightweight experiments (ossiop, ossmpip)

### **Section 3: Roadmap / Future Plans**

#### **Supplemental Information**

Command Line Interface (CLI) tutorial and examples







# Open | SpeedShop™ Performance with Open | SpeedShop



# NASA Open | SpeedShop Update | Training

# Supplemental: 1 Command Line Interface Usage











#### Command Line Interface Features

- "gdb" like tool for performance data creation and viewing
- > Same functional capabilities the graphical user interface (GUI)
  - Exception: GUI can focus on the source line corresponding to statistics
- > List metadata about your application and the OSS experiment
- Create experiments and run them
- Launch the GUI from the CLI via the "opengui" command
- View performance data from a database file
  - openss –cli –f <database filename> to launch
  - expview key command with many options
  - list list many items such as source, object files, metrics to view
  - expcompare Compare ranks, threads, processes to each other, more...
  - cviewcluster Creates groups of like performing entities to outliers.
  - cview Output the columns of data representing groups created by cviewcluster.
  - cviewinfo Output what ranks, threads, etc., are in each cview group





#### Command Line Interface Features (additional)

- > Format the performance information view to csv
  - expview -F csv
- > Selectively only view, compare, analyze by rank, thread, process
  - -r <rank number> or rank list or rank ranges
  - -t <thread number> or thread list or thread ranges
  - -p process number> or process list or process ranges
- > Selectively only view, compare, analyze by specific metrics
  - -m <metric name> or list of metrics
  - Example metrics for sampling: percent, time
  - Example metrics for tracing: time, count, percent
- > Selectively only view by specific view type options
  - -v <view type> or list of view types
  - Example view types: functions, statements, loops, linked objects
  - Example metrics for tracing: time, count, percent





#### **Command Line Interface Examples**

- ❖ openss –cli –f <database file name>
- Commands to get started
  - > expstatus
    - Gives the metadata information about the experiment
  - > expview
    - Displays the default view for the experiment
    - Use expview <experiment type>nn to see only nn lines of output
      - expview pcsamp20 shows only the top 20 time taking functions
    - -v functions: displays data based on function level granularity
    - -v statements : displays data based on statement level granularity
    - -v linkedobjects: displays data based on library level granularity
    - -v loops: displays data based on loop level granularity
    - -v calltrees: displays call paths combining like paths
    - -v calltrees, fullstack : displays all unique call paths individually
    - -m loadbalance : displays the min, max, average values across ranks, ...





#### **Command Line Interface Examples**

- ❖ Openss -cli -f <database file name>
- Commands to get started
  - expview (continued from previous page)
    - -v trace : for tracing experiments, display chronological list of events
    - -m <metric> : only display the metric(s) provided via the -m option
      - Where metric can be: time, percent, a hardware counter, (see list -v metrics)
    - -r <rank or rank list> : only display data for that rank or ranks
    - -t < thread id or list of thread ids> : only display data for that thread (s)
    - -h < host id or list of host ids> : only display data for that host or hosts
    - -F csv : display performance data in a comma separated list
  - > expcompare : compare data within the same experiment
    - -r 1 -r 2 -m time : compare rank 1 to rank 2 for metric equal time
    - -h host1 -h host2 : compare host 1 to host 2 for the default metric





#### **Command Line Interface Examples**

- openss –cli –f <database file name>
- Commands to get started
  - > list
    - -v metrics: display the data types (metric) that can be displayed via –m
    - -v src : display source files associated with experiment
    - -v obj : display object files associated with experiment
    - -v ranks : display ranks associated with experiment
    - -v hosts: display machines associated with experiment
    - -v exp: display the experiment numbers that are currently loaded
    - -v savedviews: display the commands that are cached in the database

## Viewing hwcsamp data in CLI





#### openss -cli -f smg2000-hwcsamp-1.openss

# View the default report for this hwcsamp experiment

openss>>[openss]: The restored experiment identifier is: -x 1

#### openss>>**expview**

Exc	lusive CPU time	% of CPU Time	PAPI_TOT_CYC	PAPI_FP_OPS Function (defining location)
	in seconds.			
smg	3.920000000 _residual.c,152)	44.697833523	11772604888	1198486900 hypre_SMGResidual (smg2000:
cycli	2.510000000 c_reduction.c,757	28.620296465 )	7478131309	812850606 hypre_CyclicReduction (smg2000:
	0.310000000	3.534777651	915610917	48863259 opal_progress (libopen-pal.so.0.0.0)
sem	0.300000000 i_restrict.c,125)	3.420752566	910260309	100529525 hypre_SemiRestrict (smg2000:
(libn	0.290000000 npi.so.0.0.2)	3.306727480	874155835	48509938 mca_btl_sm_component_progress

# Viewing hwcsamp data in CLI





# View the linked object (library) view for this Hardware Counter Sampling experiment

#### openss>>expview -v linkedobjects

Exclusive CPU time	% of CPU Time	PAPI_TOT_CYC	PAPI_FP_OPS LinkedObject				
in seconds.							
7.710000000	87.315968290	22748513124	2396367480 smg2000				
0.610000000	6.908267271	1789631493	126423208 libmpi.so.0.0.2				
0.310000000	3.510758777	915610917	48863259 libopen-pal.so.0.0.0				
0.20000000	2.265005663	521249939	46127342 libc-2.10.2.so				
8.830000000	100.000000000	25975005473	2617781289 Report Summary				
openss>>							





```
# View the default I/O report for this I/O experiment
openss - cli -f sweep3d.mpi-iot.openss
openss>>[openss]: The restored experiment identifier is: -x 1
openss>>expview
I/O Call
         % of Number Function (defining location)
Time(ms) Total of
       Time Calls
1.241909 90.077151
                     36 write (libpthread-2.17.so)
0.076653 5.559734
                     2 close (libpthread-2.17.so)
                     2 read (libpthread-2.17.so)
0.035452 2.571376
0.024703 1.791738
                     2 open64 (libpthread-2.17.so)
```

# View the default trace (chronological list of I/O functions calls) for this I/O experiment

#### openss>>expview -v trace

```
Start Time I/O Call % of Function
                                                          File/Path Name
                                                                              Event Call Stack Function (defining location)
            Time(ms) Total Dependent
                                                                   Identifier(s)
                  Time Return
                       Value
2014/08/17 09:25:44.368 0.012356 0.896196
                                            13 input 0:140166697882752 >>>>>>open64 (libpthread-2.17.so)
2014/08/17 09:25:44.368 0.027694 2.008679
                                                input 0:140166697882752 >>>>>>>read (libpthread-2.17.so)
2014/08/17 09:25:44.377 0.053832 3.904500
                                             0 input 0:140166697882752 >>>>>>close (libpthread-2.17.so)
2014/08/17 09:25:44.378 0.012347 0.895543
                                            13 input 0:140166697882752 >>>>>>pen64 (libpthread-2.17.so)
2014/08/17 09:25:44.378 0.007758 0.562697
                                            53 input 0:140166697882752 >>>>>>read (libpthread-2.17.so)
2014/08/17 09:25:44.378 0.022821 1.655235
                                             0 input 0:140166697882752 >>>>>>close (libpthread-2.17.so)
2014/08/17 09:25:44.378 0.037219 2.699539
                                            62 /dev/pts/1 0:140166697882752 >>>>>> write (libpthread-2.17.so)
```





# View the list of metrics (types of performance information) for this I/O experiment

#### openss>>list-v metrics

iot::average

iot::count

iot::exclusive\_details

iot::exclusive\_times

iot::inclusive\_details

iot::inclusive\_times

iot::max

iot::min

iot::nsysarg

iot::pathname

iot::retval

iot::stddev

iot::syscallno

iot::threadAverage

iot::threadMax

iot::threadMin

iot::time





# View in chronological trace order: start\_time, time, the rank:thread event occurred in.

```
openss>>expview -m start time,time,id -vtrace
 Start Time(d:h:m:s) Exclusive
                                  Event Call Stack Function (defining location)
            I/O Call
                     Identifier(s)
            Time(ms)
2014/08/17 09:25:44.368  0.012356 0:140166697882752 >>>>>>>pen64 (libpthread-2.17.so)
2014/08/17 09:25:44.368  0.027694 0:140166697882752 >>>>>>>read (libpthread-2.17.so)
2014/08/17 09:25:44.377  0.053832 0:140166697882752 >>>>>>close (libpthread-2.17.so)
014/08/17 09:25:44.378  0.012347 0:140166697882752 >>>>>>>pen64 (libpthread-2.17.so)
2014/08/17 09:25:44.378  0.007758 0:140166697882752 >>>>>>>>read (libpthread-2.17.so)
2014/08/17 09:25:44.378  0.022821 0:140166697882752 >>>>>>close (libpthread-2.17.so)
2014/08/17 09:25:44.378  0.037219 0:140166697882752 >>>>>> write (libpthread-2.17.so)
2014/08/17 09:25:44.378  0.018545 0:140166697882752 >>>>>> write (libpthread-2.17.so)
2014/08/17 09:25:44.378  0.019837 0:140166697882752 >>>>>> write (libpthread-2.17.so)
2014/08/17 09:25:44.379  0.035047 0:140166697882752 >>>>>> write (libpthread-2.17.so)
```

openss>>expview -m loadbalance





# View the load balance (max, min, and average values) across all ranks, threads, or processes.

```
Max I/O Rank Min I/O Rank Average Function (defining location)
          Call of I/O Call
  Call of
  Time Max
               Time Min
                            Time
Across
            Across
                       Across
Ranks(ms)
             Ranks(ms)
                           Ranks(ms)
1.241909
           0 1.241909
                        0 1.241909 write (libpthread-2.17.so)
0.076653
           0 0.076653
                        0 0.076653 close (libpthread-2.17.so)
0.035452
           0 0.035452
                        0 0.035452 read (libpthread-2.17.so)
0.024703
           0 0.024703
                        0 0.024703 open64 (libpthread-2.17.so)
# View data for only rank nn, in this case rank 0
openss>>expview -r 0
          % of Number Function (defining location)
I/O Call
Time(ms)
           Total
                   of
       Time Calls
1.241909 90.077151
                      36 write (libpthread-2.17.so)
                      2 close (libpthread-2.17.so)
0.076653 5.559734
0.035452 2.571376
                      2 read (libpthread-2.17.so)
```

2 open64 (libpthread-2.17.so)

0.024703 1.791738





# View the top time taking call tree in this application run. iot1 indicates see only one callstack. iot<number> shows "number" of calltrees.

```
openss>>expview -v calltrees,fullstack iot1
I/O Call
        % of Number Call Stack Function (defining location)
Time(ms)
          Total
                of
      Time Calls
            _start (sweep3d.mpi)
            > @ 562 in libc start main (libmonitor.so.0.0.0: main.c,541)
            >> libc start main (libc-2.17.so)
            >>> @ 517 in monitor main (libmonitor.so.0.0.0: main.c,492)
            >>>0x4026a2
            >>>> @ 185 in MAIN (sweep3d.mpi: driver.f,1)
            >>>>> @ 41 in inner auto (sweep3d.mpi: inner auto.f,2)
            >>>>> @ 128 in inner (sweep3d.mpi: inner.f,2)
            >>>>> gfortran st write done (libgfortran.so.3.0.0)
            >>>>>>>>6456
            >>>>>>>>>
            >>>>>>>>>>
```





# Open | SpeedShop CLI output into csv form for spreadsheet use.

Create an experiment database

```
EXE=./a.out
export
OPENSS_HWCSAMP_EVENTS="PAPI_VEC_DP,FP_COMP_OPS_EXE:SSE_FP_PACKED_
DOUBLE"
osshwcsamp "/usr/bin/srun -N 1 -n 1 $EXE "
```

Open the database file and use expview –F csv to create a csv file openss –cli –f a.out-hwcsamp.openss –expview -F csv > mydata.csv

Create csv file using a script method

```
echo "exprestore -f a.out-hwcsamp.openss" > ./cmd_file
echo "expview -F csv > mydata.csv" >> ./cmd_file
echo "exit" >> ./cmd_file
#
# Run openss utility to output CSV rows.
#
openss -batch < ./cmd_file
```

### GUI export of data to csv form





Getting Open | SpeedShop output into csv form for spreadsheet use.

#### openss –f stream.x-hwcsamp-1.openss

Go to Stats Panel Menu

Select the "Report Export Data (csv)" option

In the Dialog Box provide a meaningful name such as stream.x-hwcsamp-1.csv

File will be saved to a file you specified above





### Storing csv information into spreadsheet

- Open your spreadsheet.
- Either
  - > Select open from pulldown menu
  - > import stream.x-hwcsamp-1.csv
  - Cut and paste the contents of stream.x-hwcsamp-1.csv into the spreadsheet
- Use "Data" operation and then "Text to columns",
- Select "comma" to separate the columns.
- Save the spreadsheet file





# Plotting the performance info from the spreadsheet (Libre office instructions)

- Open your spreadsheet.
- Select Insert, then choose Chart
- Choose a chart type
- Choose data range choices (data series in columns)
- Choose data series (nothing to do here)
- Chart is created. Views can be changed.







# Open SpeedShop™



# Performance with Open | SpeedShop

# NASA Open | SpeedShop Update / Training

# Supplemental: 2 Preferences





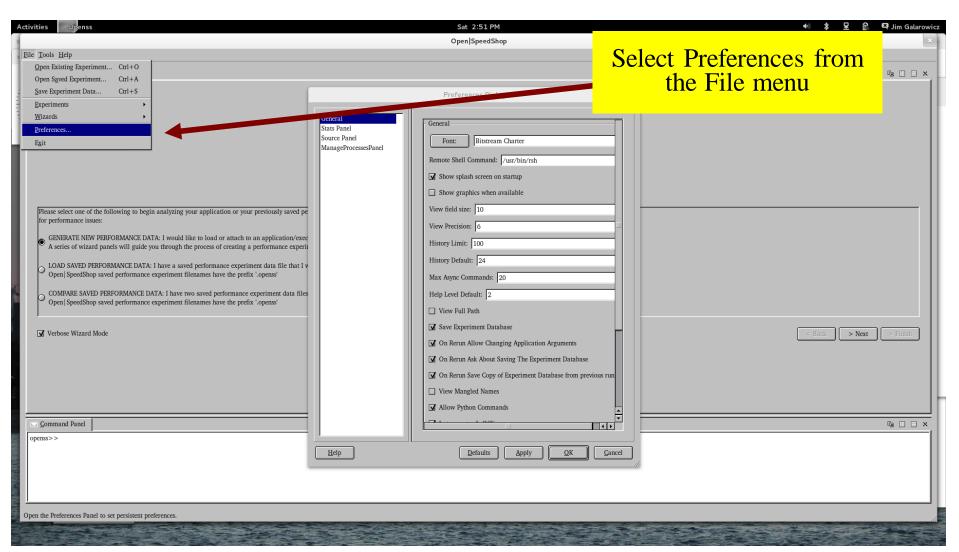


# Changing Preferences in O|SS





### Selecting File->Preferences to customize O | SS



# Changing Preferences in O|SS





Enabling the new Command Line Interface (CLI) save/reuse views feature. Looking for friendly user

evaluation.

