Open | SpeedShop™

How to use Open|SpeedShop to Analyze the Performance of Parallel Codes.

Tutorial @ NASA
October 5, 2011

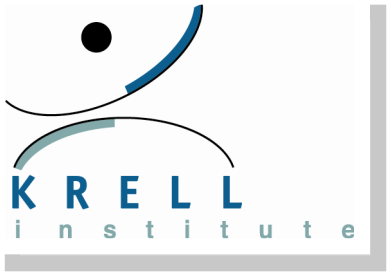
Jim Galarowicz, Krell Institute

Don Maghrak, Krell Institute

Tutorial Outline

- **Welcome, Introductions, Performance Analysis Motivations**
- **① Running first experiment**
- **② Sampling experiments:**
  - pcsamp, usertime and hwc/hwctime/hwcsamp
- **③ I/O Tracing Experiments**
- **④ How to Detect Bottlenecks in Parallel Codes?**
- **⑤ User Interfaces**
- **⑥ Comparing Experiment Results**
- **⑦ Static binary support (focused for Cray and BG/P)**
- **⑧ Other projects related to Open|SpeedShop**
- **⑨ Module Files & Running OSS on NASA systems**
- **Conclusions**

OSS Development

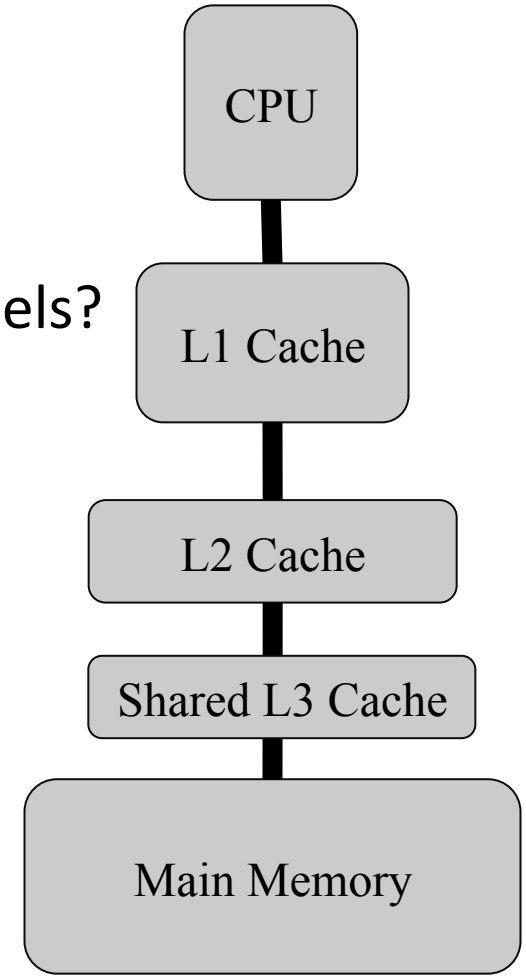


- **Jim Galarowicz, Krell**
- **Don Maghrak, Krell**

- **Tri-Lab ASC CCE Management**
  - Martin Schulz, LLNL
  - David Montoya, LANL
  - Mahesh Rajan, SNLs
- **Larger team**
  - William Hachfeld and Dave Whitney, Krell
  - Dane Gardner, LANL
  - Scott Cranford and Joseph Kenny, SNLs
  - Chris Chambreau and Matt Legendre, LLNL
  - Dyninst group (Bart Miller, UW & Jeff Hollingsworth, UMD)
  - Phil Roth, ORNL
  - Ciera Jaspan, CMU

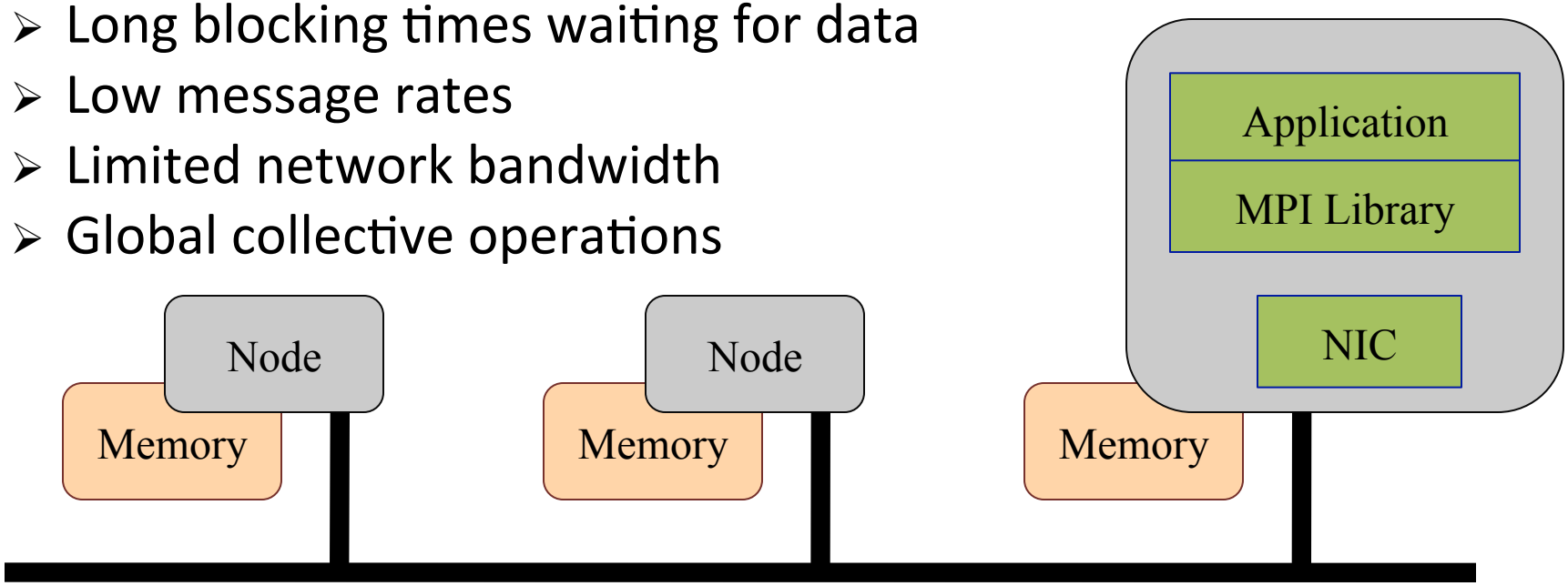
What to Look For: Sequential Runs

- **Step 1: Identify computational intensive parts**
  - Where am I spending my time?
    - Modules/Libraries
    - Statements
    - Functions
  - Is the time spent in the computational kernels?
  - Does this match my intuition?
- **Impact of memory hierarchy**
  - Do I have excessive cache misses?
  - How is my data locality?
  - Impact of TLB misses?
- **External resources**
  - Is my I/O efficient?
  - Time spent in system libraries?



What to Look For: Message Passing

- **Distributed Memory Model**
  - Sequential/shared memory nodes coupled by a network
  - Only local memory access
  - Data exchange using message passing (e.g., MPI)
- **Typical performance issues**
  - Long blocking times waiting for data
  - Low message rates
  - Limited network bandwidth
  - Global collective operations



Performance Tool Terminology

- **Statistical Sampling**
  - Periodically interrupt execution and record location
  - Report statistical distribution across all reported locations
  - Data typically aggregated over time
  - Most common metric is time, but other metrics possible
  - Useful to get an overview
  - Low and uniform overhead
- **Event Tracing**
  - Gather and store individual application events
  - Examples: function invocations, MPI messages, I/O calls
  - Events are typically time stamped
  - Provides detailed per event information
  - Can lead to huge data volumes
  - Higher overhead, potentially in bursts

Open | SpeedShop™

Section 1
Running First Experiment

Tutorial @ NASA
October 5, 2011

How to use Open|SpeedShop to Analyze the Performance of Parallel Codes.

Experiment Workflow

Open|SpeedShop Workflow

Application

"Experiment"

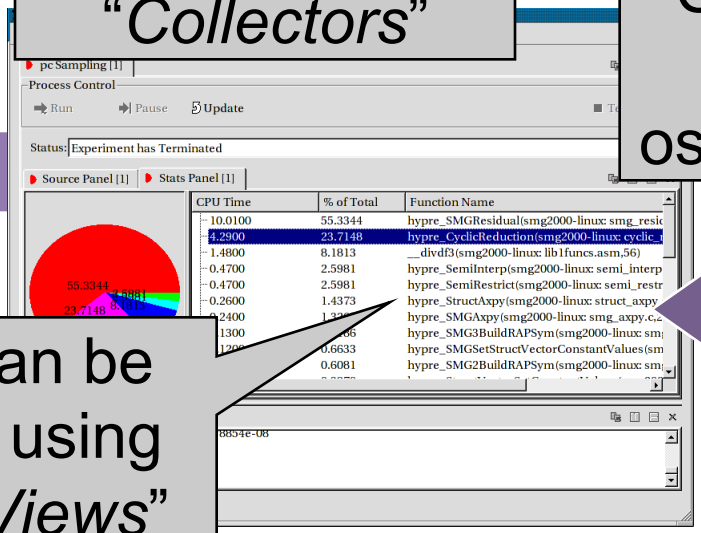
Open | SpeedShop™

Run

Consists of one or more data "*Collectors*"

Convenience Scripts osspcsamp, etc.

View

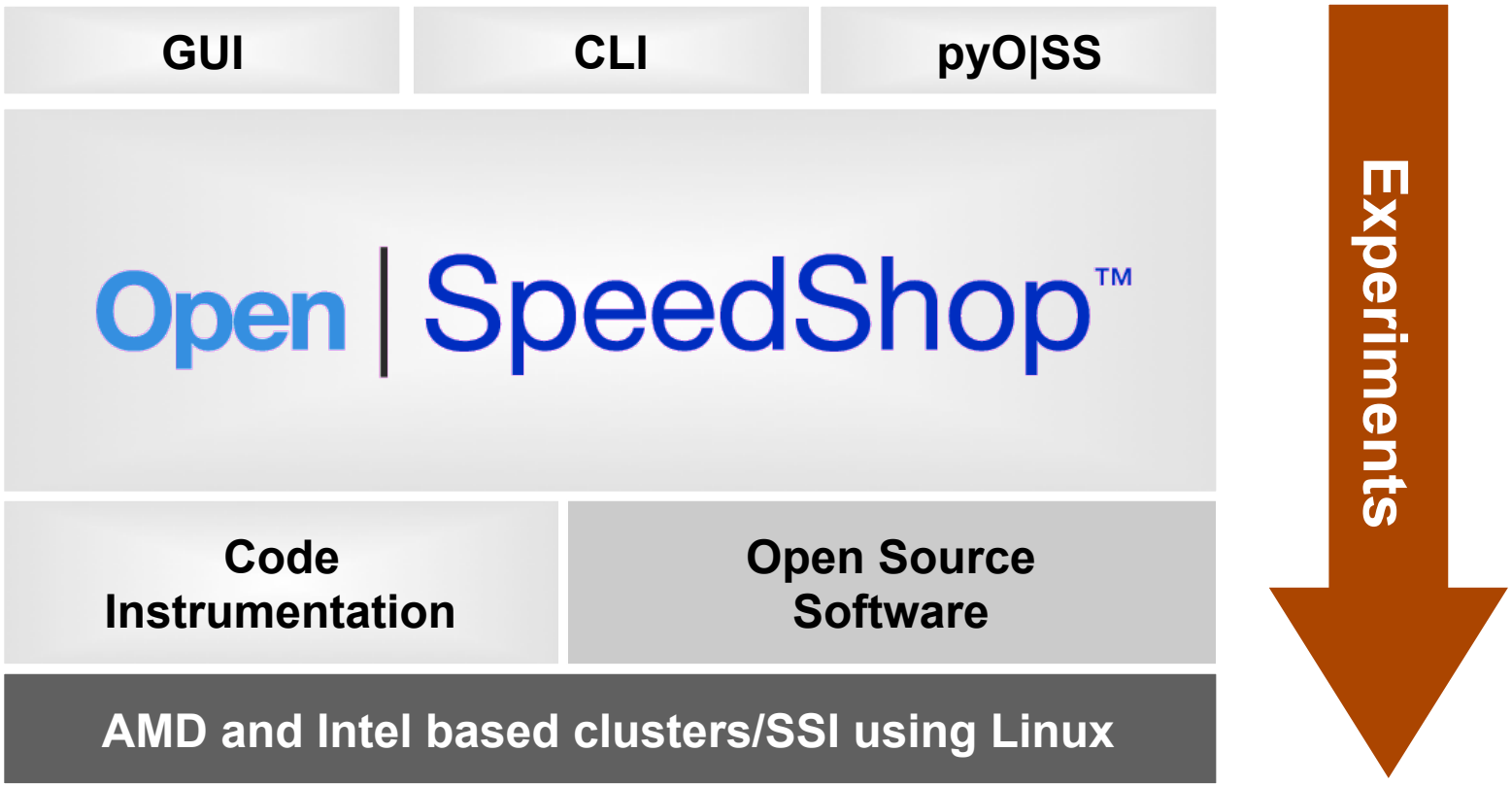


Results

Results can be displayed using several "*Views*"

Stored in SQL database

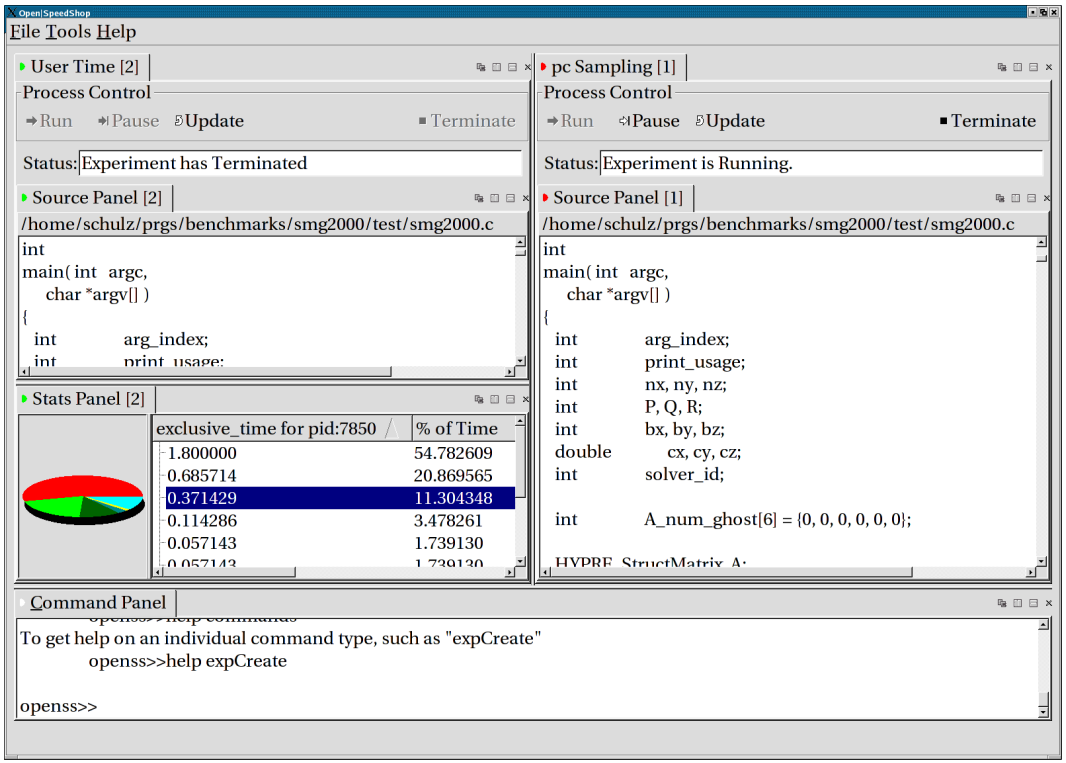
High-level Architecture



Open Source: PAPI, libmonitor, sqlite, Dyninst, MRNet, libdwarf, libelf, python, etc..

Basic Interface Usage

- **Step 1**
  - Gather data from command line
  - Example: osspcsamp “<application>”
  - Create database
- **Step 2**
  - Analyze data in GUI
  - Simple graphics
  - Relate data to source
  - openss –f <db file>



Performance Experiments

- **Concept of an Experiment**
  - What to measure and what to analyze?
  - Experiment type is chosen by user (type of data to gather)
  - Any experiment can be applied to any application
- **Consists of Collectors and Views**
  - Collectors define specific performance data sources
    - Program counter samples
    - Call stack samples
    - Hardware counters
    - Tracing of library routines (I/O, MPI)
  - Views specify performance data aggregation and presentation
  - Multiple collectors per experiment possible (online version)

Sampling Experiments

- **PC Sampling (pcsamp)**
  - Record PC in user defined time intervals
  - Low overhead overview of time distribution
  - Good first step, lightweight overview
- **Call Path Profiling (usertime)**
  - PC Sampling and Call stacks for each sample
  - Provides inclusive and exclusive timing data
  - Use to find hot call paths, whom is calling who
- **Hardware Counters (hwc, hwctime, hwcsamp)**
  - Access to data like cache and TLB misses
  - hwc, hwctime:
    - Sample a HWC event based on an event threshold
    - Default event is PAPI_TOT_CYC overflows
  - hwcsamp:
    - Sample up to six events based on a sample time (hwcsamp)
    - Default events are PAPI_FP_OPS and PAPI_TOT_CYC

Tracing Experiments

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

Running a First Experiment

1. **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 256 smg2000 –n 65 65 65"
3. **Storing the results**
   - O|SS will create a database
   - Name: smg2000-pcsamp.openss
4. **Exploring the gathered data**
   - O|SS will print a default report
   - Open the GUI to analyze data in detail (run: "**openss**")

Example Run with Output

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

```
osspcsamp "mpirun -np 2 ./smg2000 -n 65 65 65"
[openss]: pcsamp experiment using the pcsamp experiment default sampling rate: "100".
[openss]: Using OPENSS_PREFIX installed in /opt/OSS-mrnet
[openss]: Setting up offline raw data directory in /tmp/jeg/offline-oss
[openss]: Running offline pcsamp experiment using the command:
"mpirun -np 2 /opt/OSS-mrnet/bin/ossrun "./smg2000 -n 65 65 65" pcsamp"

Running with these driver parameters:
 (nx, ny, nz)    = (65, 65, 65)
 (Px, Py, Pz)    = (2, 1, 1)
 (bx, by, bz)    = (1, 1, 1)
 (cx, cy, cz)    = (1.000000, 1.000000, 1.000000)
 (n_pre, n_post) = (1, 1)
 dim             = 3
 solver ID       = 0
=============================================
Struct Interface:
=============================================
Struct Interface:
 wall clock time = 0.049847 seconds
 cpu clock time  = 0.050000 seconds
```

Example Run with Output

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

```
=============================================
Setup phase times:
=============================================
SMG Setup:
 wall clock time = 0.635208 seconds
 cpu clock time  = 0.630000 seconds
=============================================
Solve phase times:
=============================================
SMG Solve:
 wall clock time = 3.987212 seconds
 cpu clock time  = 3.970000 seconds
Iterations = 7
Final Relative Residual Norm = 1.774415e-07
[openss]: Converting raw data from /tmp/jeg/offline-oss into temp file X.0.openss

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

Example Run with Output

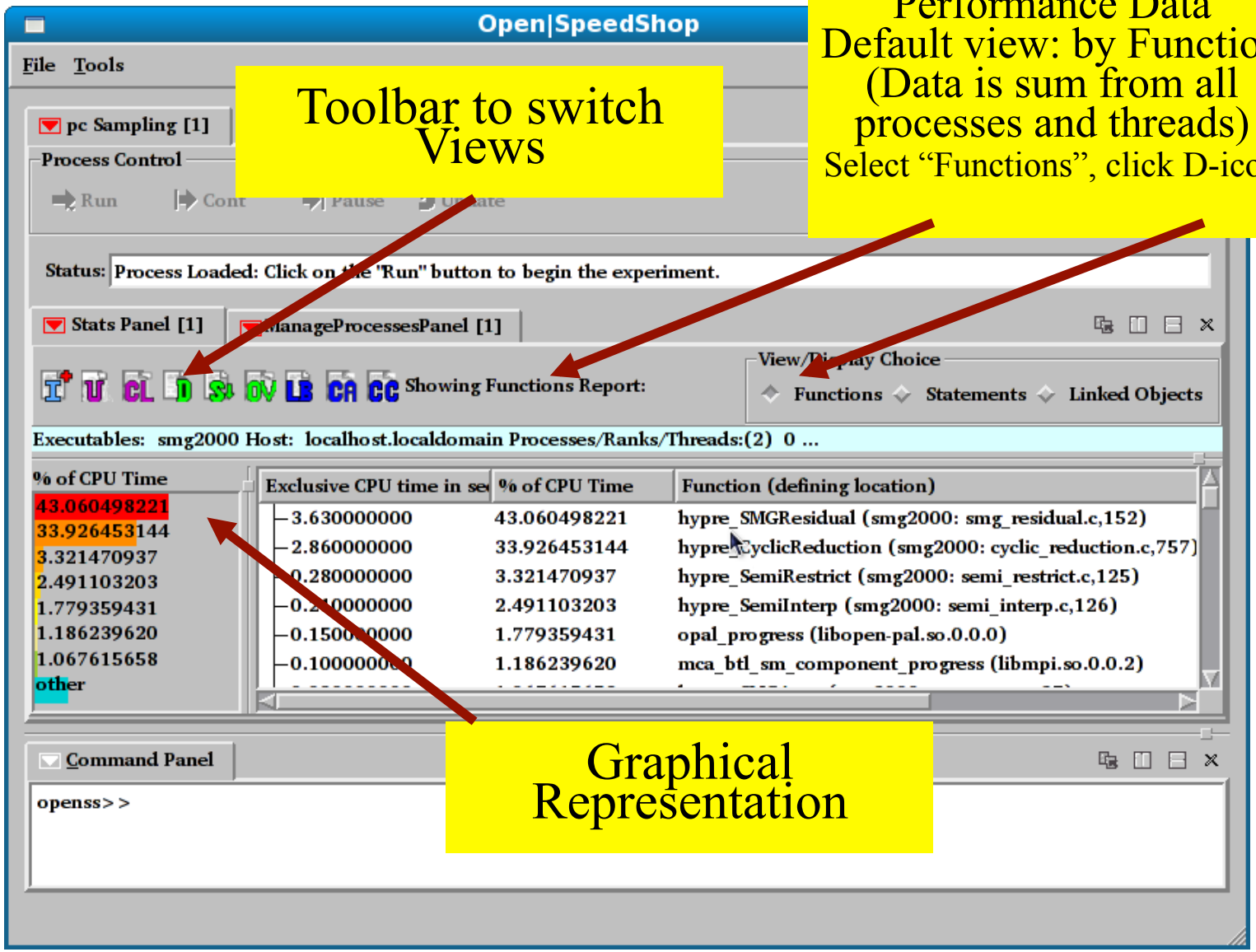
- **osspcsamp "mpirun –np 2 smg2000 –n 65 65 65"**

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

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

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

Default Output Report View

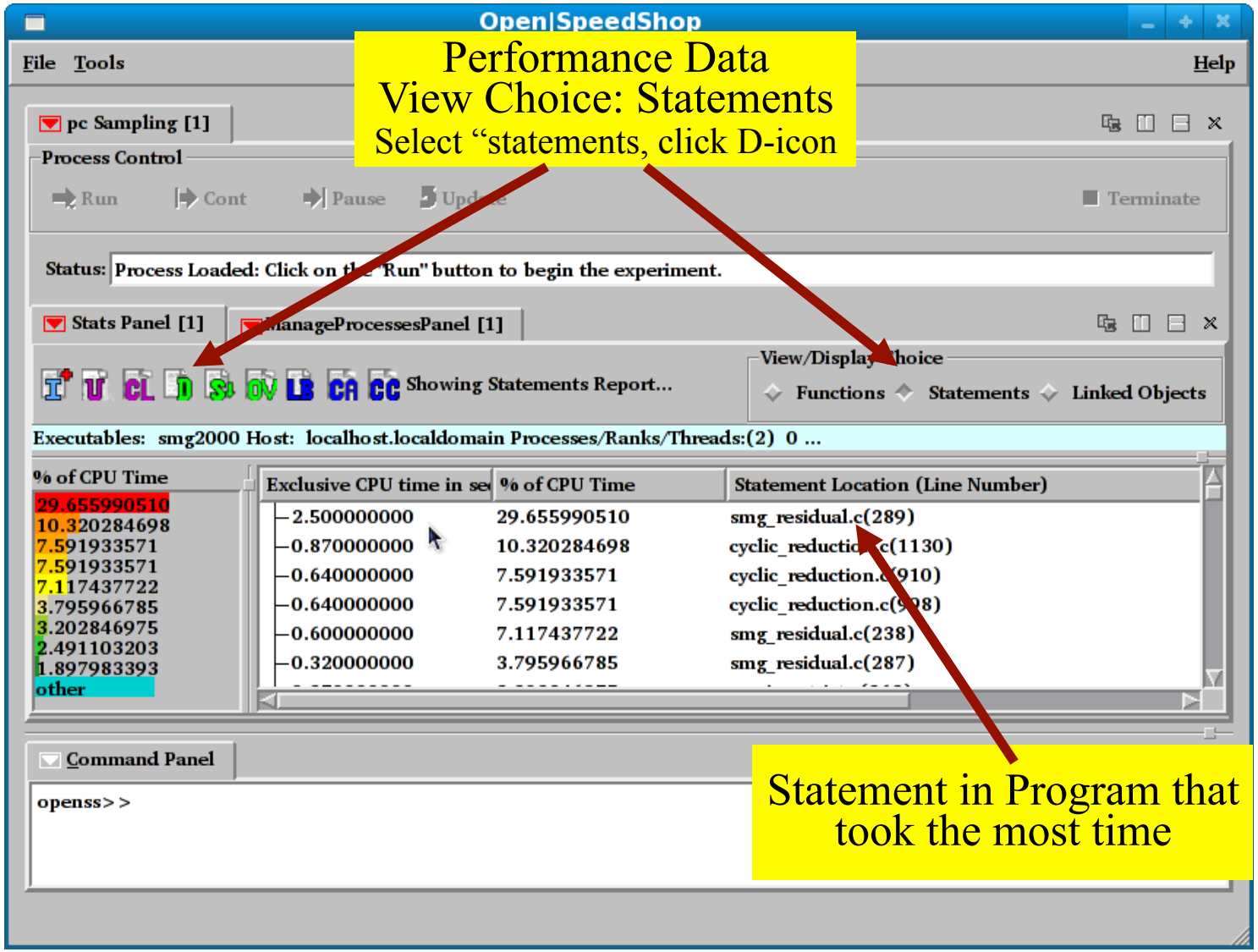


Toolbar to switch Views

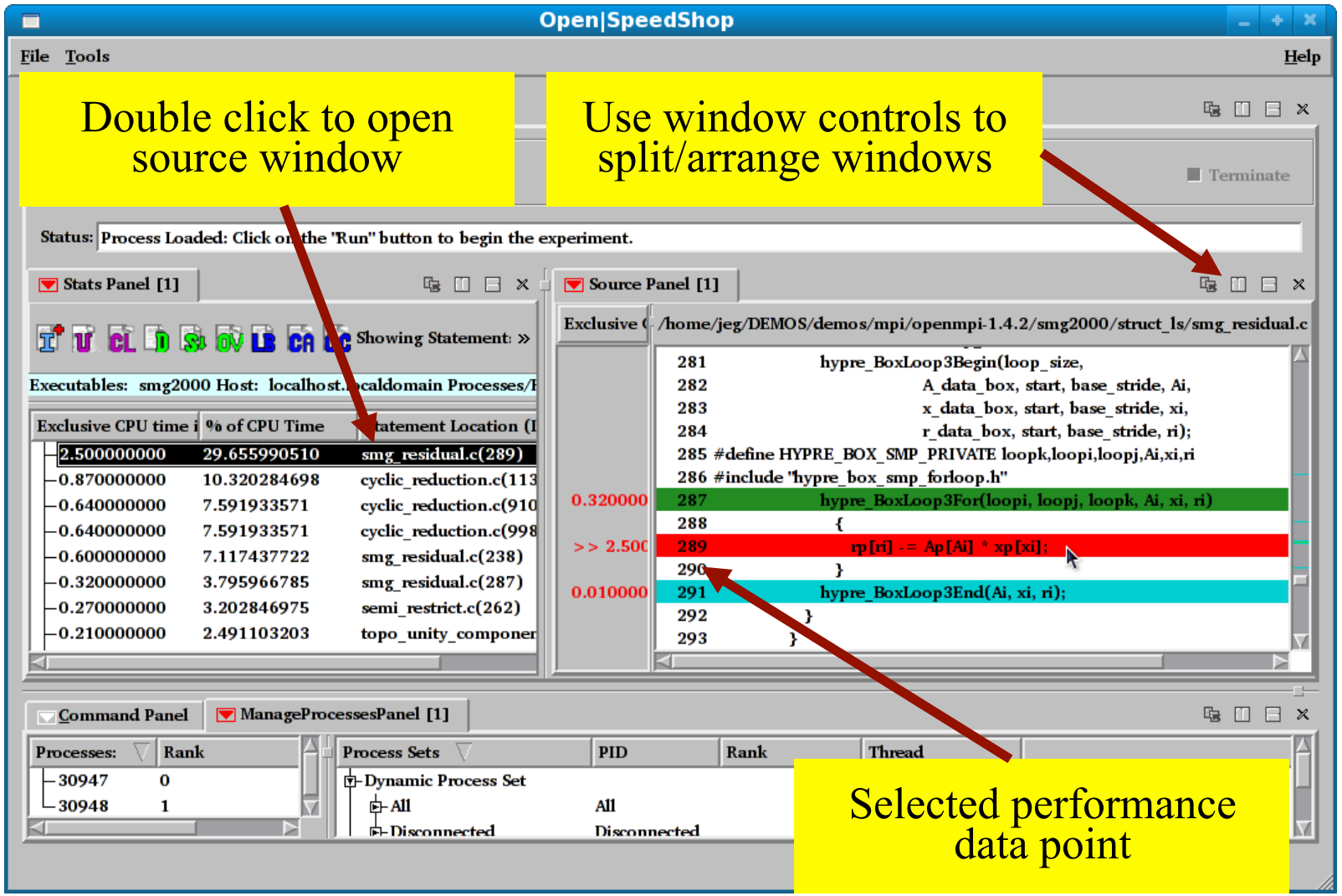
Performance Data Default view: by Function (Data is sum from all processes and threads)
Select "Functions", click D-icon

Graphical Representation

Statement Report Output View



Associate Source & Performance Data



Take away messages from this section

- **The take away messages are:**
  - Place the way you **run** your application **normally, but** in quotes and pass it as an argument to osspcsamp, or any of the other experiment convenience scripts: ossio, ossmpi, etc.
    - **osspcsamp** "srun –N 8 –n 64 ./mpi_application app_args"
  - Open|SpeedShop outputs default view 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**
    - Need a shared file system across all nodes of a cluster
    - **export OPENSS_RAWDATA_DIR=**/shared/user
  - Start with pcsamp for overview of performance
  - Then "zero" into performance issues with other experiments, if necessary

Open | SpeedShop™

Section 2
Sampling Experiments

Tutorial @ NASA
October 5, 2011

How to use Open|SpeedShop to Analyze the Performance of Parallel Codes.

Sampling Experiments in O|SS

- **PC Sampling**
  - Approximates CPU Time for Line and Function
  - No Call Stacks
  - Convenience Script: osspcsamp
- **User Time**
  - Inclusive vs. Exclusive CPU Time
  - Includes Call Stacks
  - Convenience Script: ossusertime
- **HW Counters**
  - Samples Hardware Counter Overflows (hwc, hwctime)
  - Periodically samples up to six (6) Hardware Counter Events (hwcsamp)
  - Convenience Scripts: osshwc, osshwctime, osshwcsamp

Running Program Counter Sampling

Offline pcsamp experiment on smg2000 application

Convenience script basic syntax:

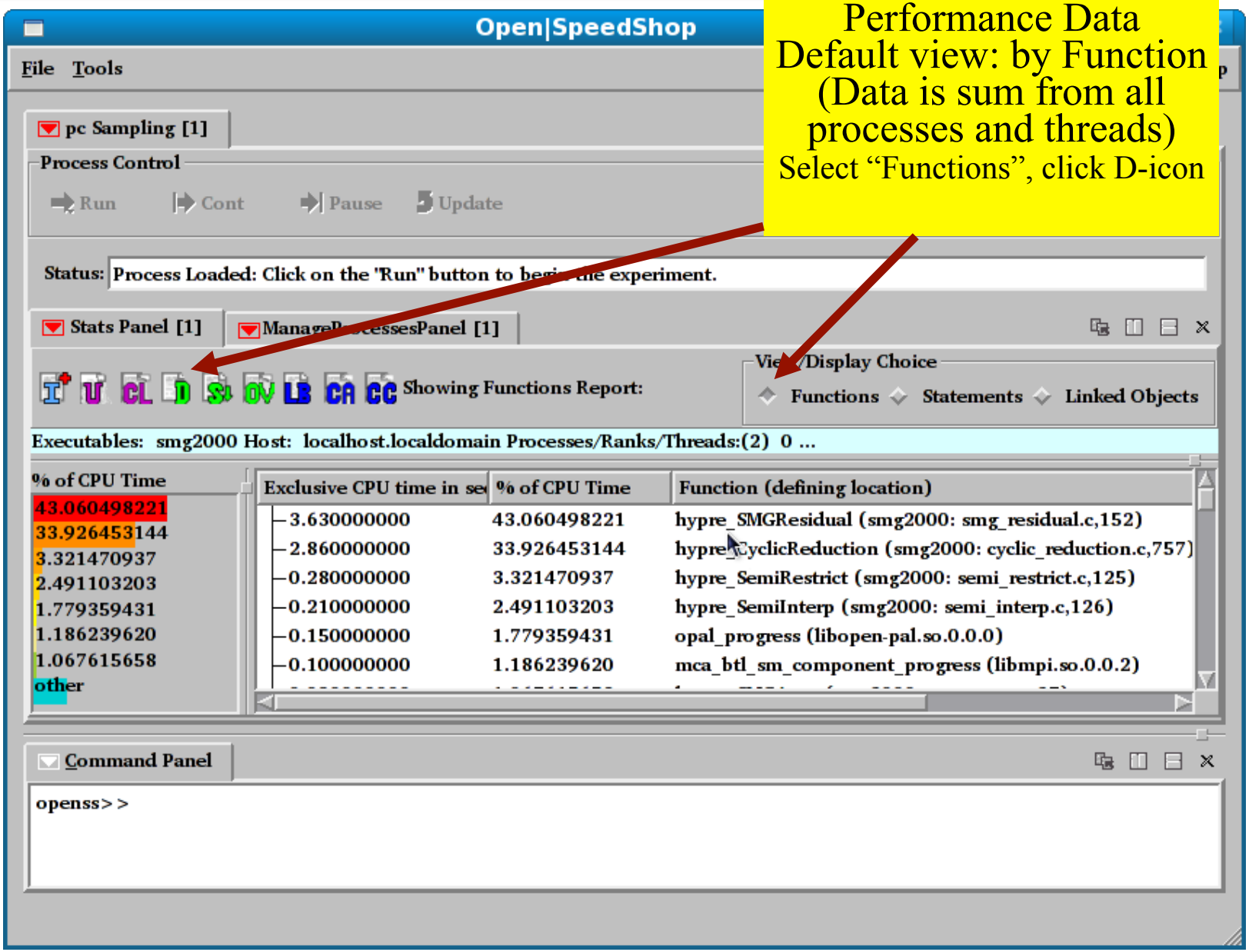
osspcsamp "smg2000 –n 50 50 50"

osspcsamp "smg2000 –n 50 50 50" **high**

- Optional Parameter:
  - Sampling frequency (samples per second)
  - Alternative parameter: **high** (200) | **low** (50) | **default** (100)

Recommendation: compile code with –g to get statements!

Viewing Flat Profiles

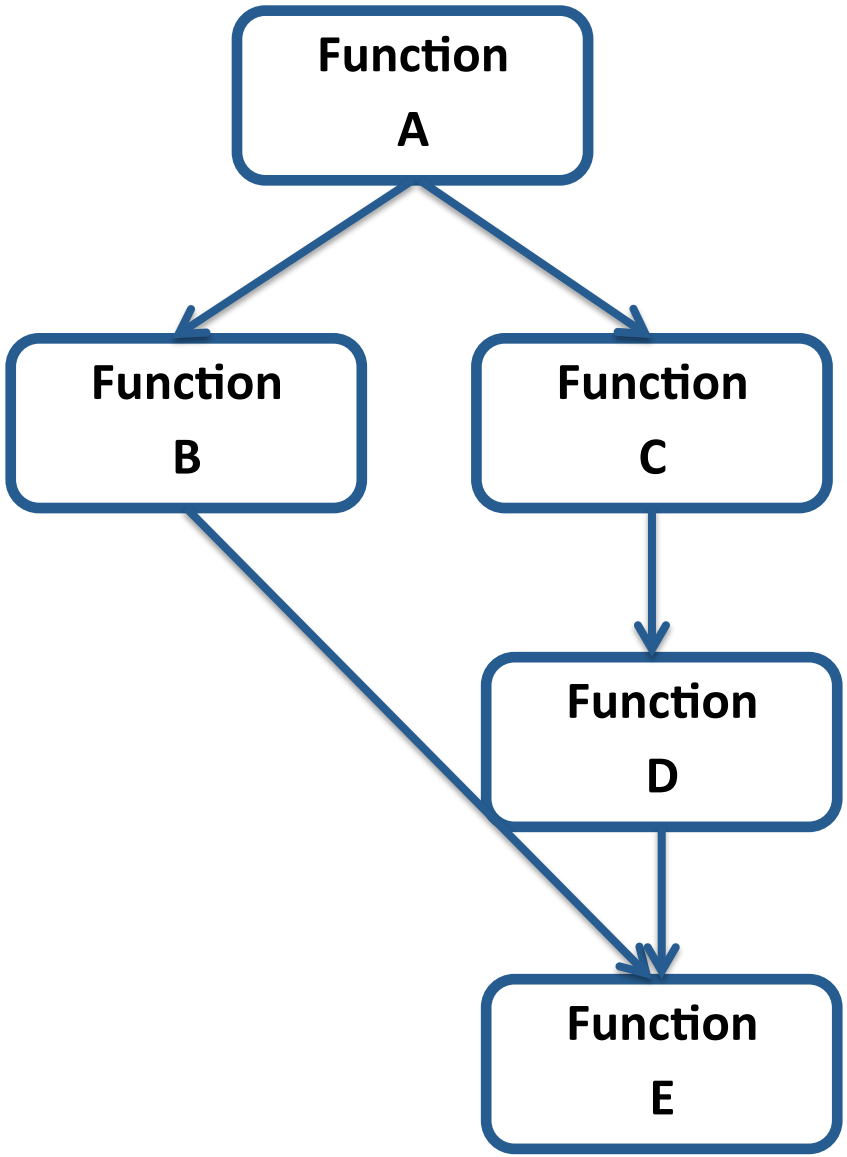


Performance Data
Default view: by Function
(Data is sum from all processes and threads)
Select "Functions", click D-icon

Identifying Critical Regions

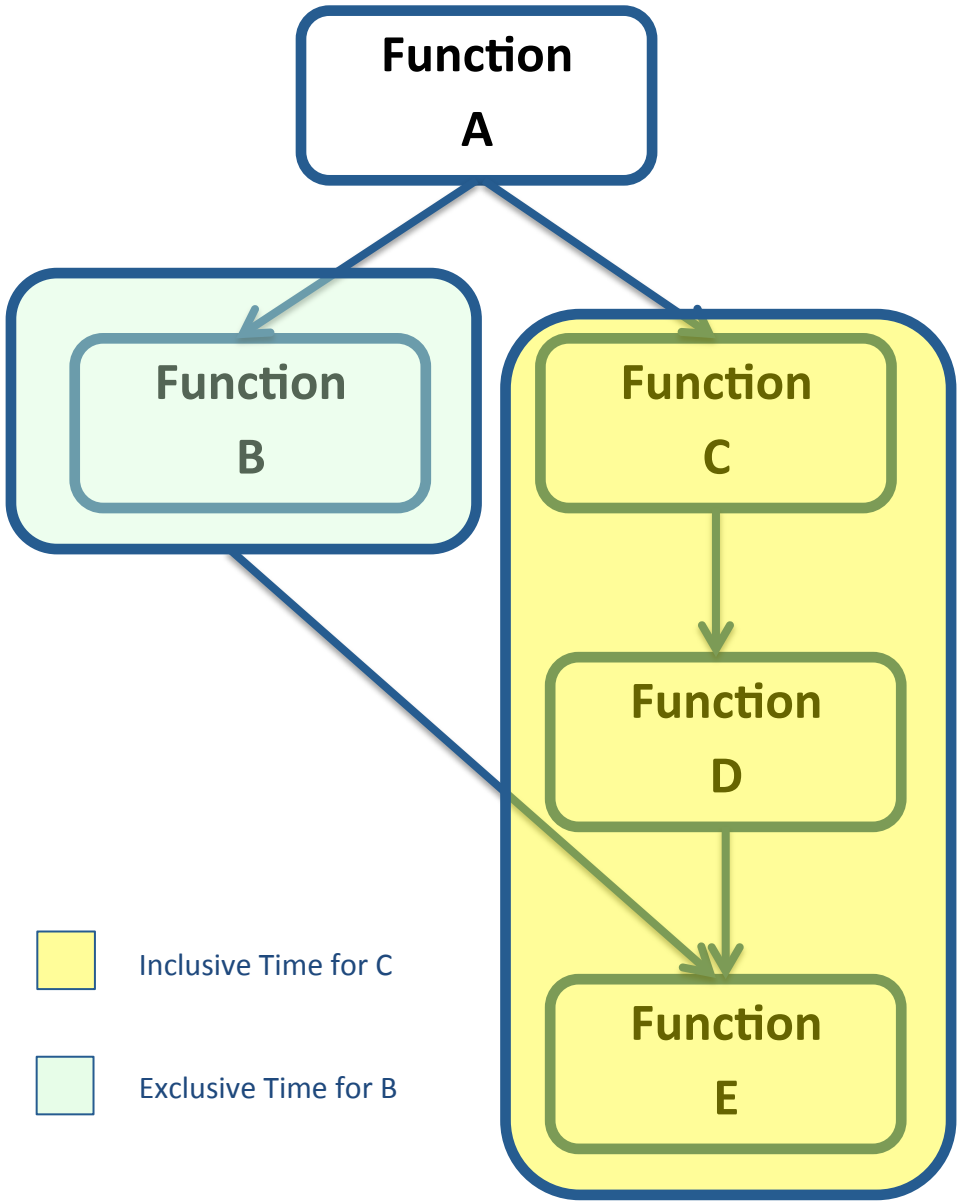
- **Profiles show computationally intensive code regions**
  - First views: Time spent per functions or per statements
    - Default view reports statistics based on functions
    - For statements, choose Statements View option
- **Questions:**
  - Are those functions/statements expected?
  - Do they match the computational kernels?
  - Any runtime functions taking a lot of time?
- **Identify bottleneck components**
  - View the profile aggregated by shared objects
    - Linked Object View option
  - Correct/expected modules?
  - Impact of support and runtime libraries

Adding Context (usertime experiment)



- **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 (shows caller/ callee relationships)
  - Hot call paths
    - Paths through application that take most time

Adding Context (usertime experiment)



- **Provides inclusive/exclusive time**
  - Time spent inside a function only (exclusive)
    - See: Function B
  - Time spent inside a function and its children (inclusive)
    - See Function C and children
- **Similar to the pcsamp experiment**
  - Collect program counter information
  - But also: collect call stack information at every sample
- **Tradeoffs**
  - Obtain additional context information
  - Have higher overhead/lower sampling rate

Running Usertime Experiments

Offline usertime experiment on smg2000 application

Convenience script basic syntax:

ossusertime "smg2000 –n 50 50 50"

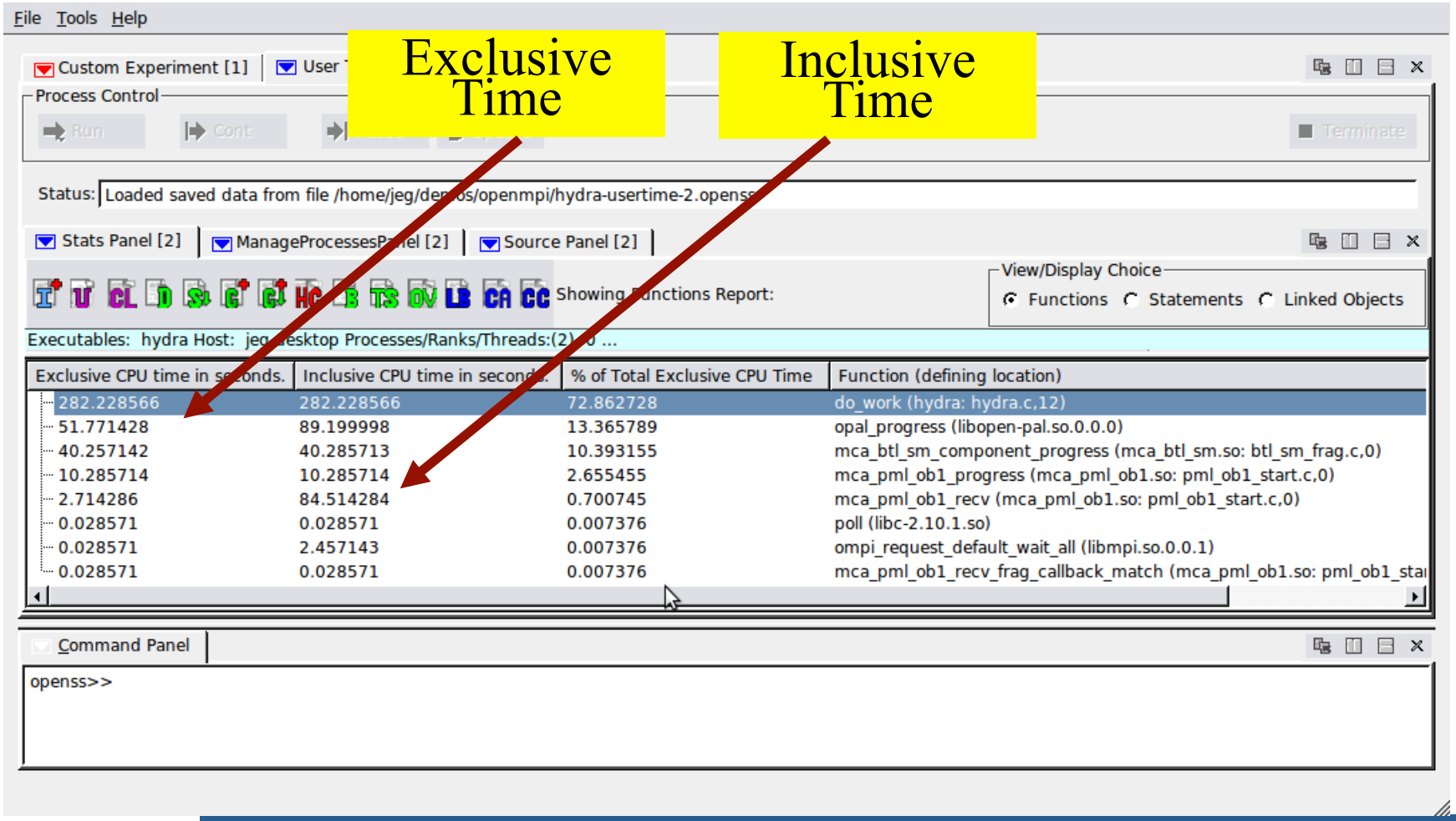
ossusertime "smg2000 –n 50 50 50" **low**

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

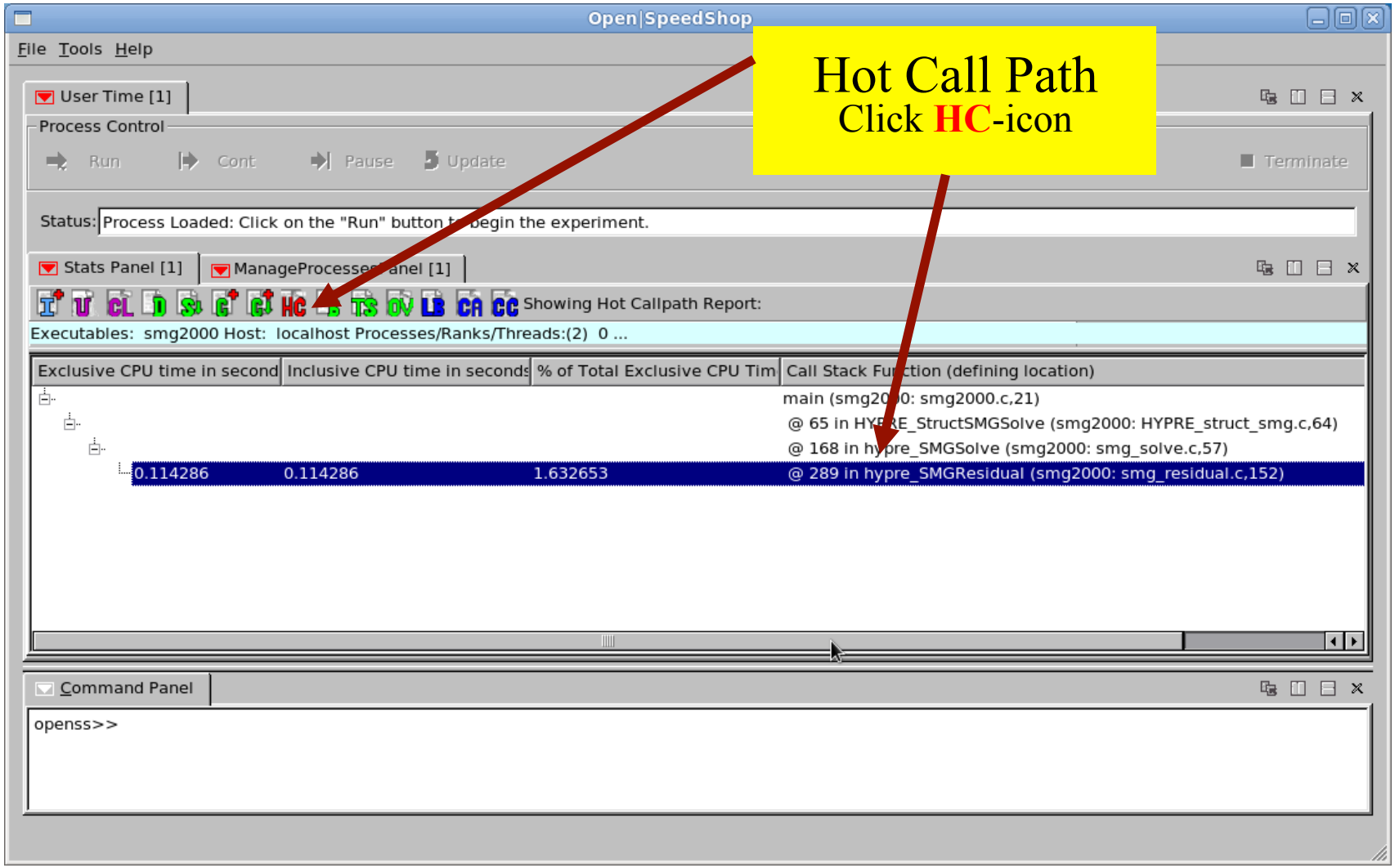
Recommendation: compile code with –g to get statements!

Viewing the Usertime Experiment

- **Default View**
  - Similar to pcsamp view from first example
  - Calculates inclusive versus exclusive times



Stack Trace Views: Hot Call Path



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: Thresholding**
  - User selects counter
  - Run until a fixed number of events have been reached
  - Take PC sample at that location
  - Reset number of events
  - Ideal number of events (threshold) depends on application
- **Basic model 2: Timer Based Sampling**
- **Three versions**
  - HWC = flat hardware counter profile (threadholding)
  - HWCtime = profiles with context / stacktraces (thresholding)
  - HWCsamp = PC sampling with multiple HWCs (timer based)

Running hwc and hwctime

Example: Offline hwc[time] experiment on SMG2000

Convenience Script Basic Syntax

osshwc[**time**] "smg2000" **<counter> <threshold>**

Where **<event>**: A PAPI event, by default PAPI_TOT_CYC

and/or **<threshold>**: Numeric value for threshold value

Default is: 10000 events before a sample is recorded

NOTE: If no data is generated, try lowering the threshold value

Available counters:

- Any counter reported by "papi_avail –a" and not derived
  - Or, run osshwc or osshwctime without arguments
- Native counters listed in the PAPI documentation

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: Not all counters exist on all platforms (check with `papi_avail`)

Running hwcsamp

Example: Offline hwcsamp experiment on SMG2000

Convenience Script Basic Syntax:

osshwcsamp "smg2000" **<event list> <sampling rate>**

Where **<event list>**: Comma separated list of PAPI events
: Default is: PAPI_TOT_CYC, PAPI_FP_OPS

And **<sampling rate>**: Numeric value for samples per second
Default 100 samples per second

Available counter

- Any counters reported by "papi_avail"
- Native counters listed in the PAPI documentation
  - Native counters also reported by "papi_native_avail"

Possible HWC Combinations To Use*

Xeons

- **PAPI_FP_INS,PAPI_LD_INS,PAPI_SR_INS (load store info, memory bandwidth needs)**
- **PAPI_L1_DCM,PAPI_L1_TCA (L1 cache hit/miss ratios)**
- **PAPI_L2_DCM,PAPI_L2_TCA (L2 cache hit/miss ratios)**
- **LAST_LEVEL_CACHE_MISSES,LAST_LEVEL_CACHE_REFERENCES (L3 cache info)**
- **MEM_UNCORE_RETIRED:REMOTE_DRAM,MEM_UNCORE_RETIRED:LOCAL_DRAM (local/nonlocal memory access)**

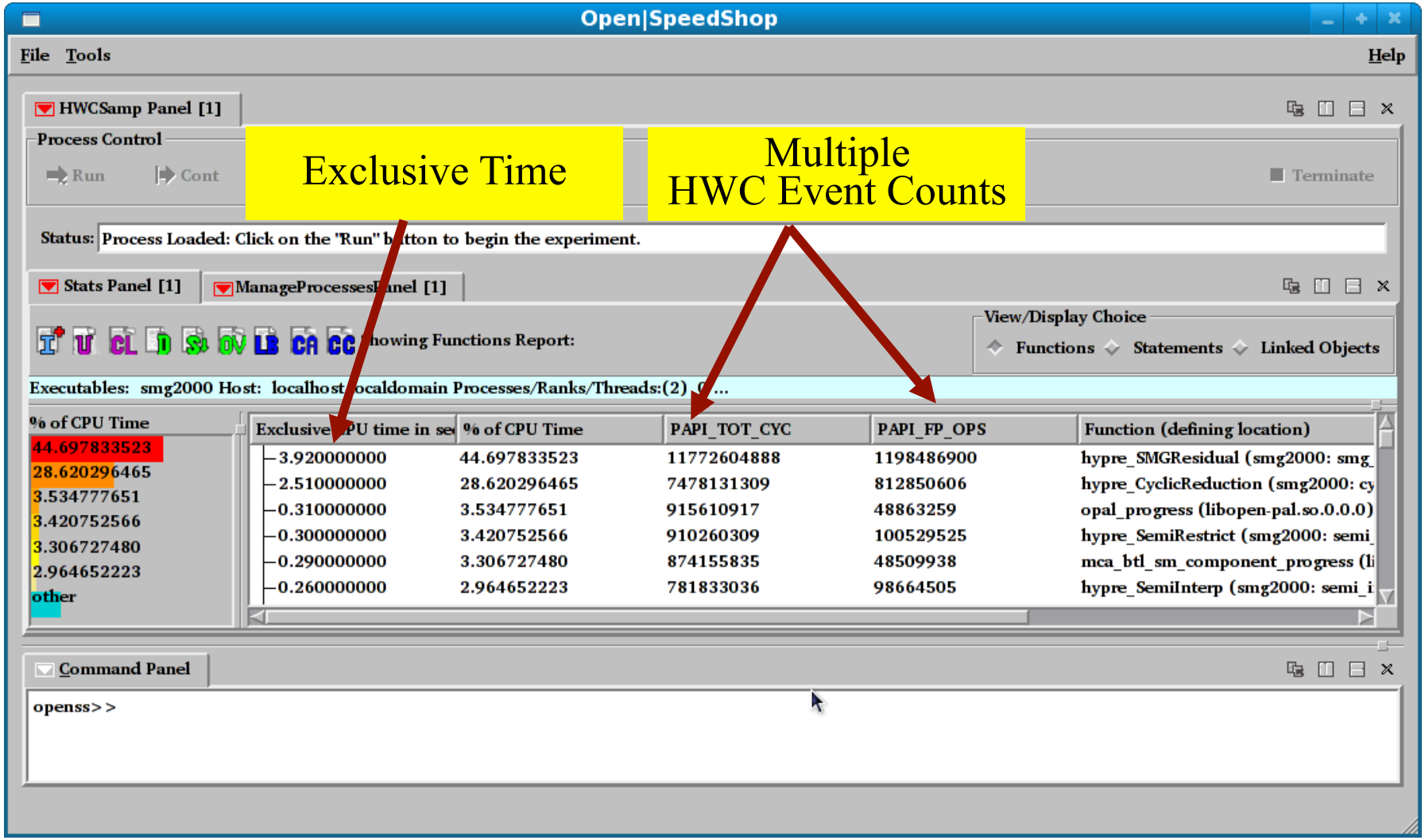
For Opterons only

- **PAPI_FAD_INS,PAPI_FML_INS (Floating point add mult)**
- **PAPI_FDV_INS,PAPI_FSQ_INS (sqrt and divisions)**
- **PAPI_FP_OPS,PAPI_VEC_INS (floating point and vector instructions)**
- **READ_REQUEST_TO_L3_CACHE:ALL_CORES,L3_CACHE_MISSES:ALL_CORES (L3 cache)**

*Credit: Koushik Ghosh, LLNL

Viewing hwcsamp Data

- **hwcsamp default view: Counter = Total Cycles & FP ops**



Viewing hwcsamp data in CLI

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

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

openss>>**expview**

Exclusive CPU time in seconds.	% of CPU Time	PAPI_TOT_CYC	PAPI_FP_OPS	Function (defining location)
3.920000000	44.697833523	11772604888	1198486900	hypre_SMGResidual (smg2000: smg_residual.c,152)
2.510000000	28.620296465	7478131309	812850606	hypre_CyclicReduction (smg2000: cyclic_reduction.c,757)
0.310000000	3.534777651	915610917	48863259	opal_progress (libopen-pal.so.0.0.0)
0.300000000	3.420752566	910260309	100529525	hypre_SemiRestrict (smg2000: semi_restrict.c,125)
0.290000000	3.306727480	874155835	48509938	mca_btl_sm_component_progress (libmpi.so.0.0.2)

Viewing hwcsamp data in CLI

openss>>expview -v linkedobjects

Exclusive CPU time in seconds.	% of CPU Time	PAPI_TOT_CYC	PAPI_FP_OPS	LinkedObject
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
0.200000000	2.265005663	521249939	46127342	libc-2.10.2.so
8.830000000	100.000000000	25975005473	2617781289	Report Summary

openss>>**expview -m flops**

Mflops	Function (defining location)
478.639000000	hypre_ExchangeLocalData (smg2000: communication.c,708)
456.405900000	hypre_StructAxpy (smg2000: struct_axpy.c,25)
452.758600000	mca_pml_ob1_isend (libmpi.so.0.0.2)
447.734300000	MPI_Irecv (libmpi.so.0.0.2)

Note about derived metrics: Working on ability to do arithmetic on hwc event counts to create user derived metrics.

Open | SpeedShop™

Section 3
I/O Tracing

Tutorial @ NASA

October 5, 2011

How to use Open|SpeedShop to Analyze the Performance of Parallel Codes.

Need for Understanding I/O

- **I/O could be significant percentage of execution time dependent upon:**
  - Checkpoint, analysis output, and visualization & I/O frequencies
  - I/O pattern in the application:
    N-to-1, N-to-N; simultaneous writes or requests
  - Nature of application:
    data intensive, traditional HPC, out-of-core
  - File system and Striping: NFS, Lustre, Panasas, and #of OSTs
  - I/O libraries: MPI-IO, hdf5, PLFS,…
  - Other jobs stressing the I/O sub-systems
- **Obvious candidates to explore first while tuning:**
  - Use parallel file system
  - Optimize for I/O pattern
  - Match checkpoint I/O frequency to MTBI of the system
  - Use appropriate libraries

Running I/O Experiments

Offline io/iot experiment on smg2000 application

Convenience script basic syntax

ossio[t] “smg2000 –n 50 50 50” [default | <list of I/O func>]

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

Additional I/O analysis with O|SS

- **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.

Additional I/O analysis with O|SS

```
openss>>expview
 I/O Call Time(ms)     % of Total Time     Number of Calls Function (defining location)
      17.906473000       96.239580457                2 __libc_read (libpthread-2.10.2.so)
       0.394533000        2.120444958               36 __libc_write (libpthread-2.10.2.so)
       0.265676000        1.427894079               72 llseek (libpthread-2.10.2.so)
       0.024794000        0.133257072                2 __libc_close (libpthread-2.10.2.so)
       0.014666000        0.078823434                2 __libc_open (libpthread-2.10.2.so)


openss>>expview -vtrace
      Start Time     I/O Call Time(ms)      % of Total Time Function Dependent       File/Path Name  Call Stack Function (defining location)
 Return Value
2011/04/22 15:28:49         0.009847000         0.052923384           10              input  >>>>>>>>>>>__libc_open (libpthread-2.10.2.so)
2011/04/22 15:28:49         0.003003000         0.016139832            0  /home/jeg/DEMOS/demos/mpi/openmpi-1.4.2/sweep3d/input
>>>>>>>>>>llseek (libpthread-2.10.2.so)
2011/04/22 15:28:49        17.902981000        96.220812461           53  /home/jeg/DEMOS/demos/mpi/openmpi-1.4.2/sweep3d/input
>>>>>>>>>>>>>__libc_read (libpthread-2.10.2.so)
2011/04/22 15:28:49         0.013479000         0.072443820            0  /home/jeg/DEMOS/demos/mpi/openmpi-1.4.2/sweep3d/input
>>>>>>>>>__libc_close (libpthread-2.10.2.so)
2011/04/22 15:28:49         0.004819000         0.025900050           10              input  >>>>>>>>>>>__libc_open (libpthread-2.10.2.so)
2011/04/22 15:28:49         0.002095000         0.011259723            0  /home/jeg/DEMOS/demos/mpi/openmpi-1.4.2/sweep3d/input
>>>>>>>>>>llseek (libpthread-2.10.2.so)
```

I/O output via CLI (equivalent of HC in GUI)

```
openss>>expview -vcalltrees,fullstack iot1
 I/O Call Time(ms)      % of Total Time      Number of Calls  Call Stack Function (defining location)
                                                              _start (sweep3d.mpi)
                                                               > @ 470 in __libc_start_main (libmonitor.so.0.0.0: main.c,450)
                                                                >>__libc_start_main (libc-2.10.2.so)
                                                                 >>> @ 428 in monitor_main (libmonitor.so.0.0.0: main.c,412)
                                                                  >>>>main (sweep3d.mpi)
                                                                   >>>>> @ 58 in MAIN__ (sweep3d.mpi: driver.f,1)
                                                                    >>>>>> @ 25 in task_init_ (sweep3d.mpi: mpi_stuff.f,1)
                                                                     >>>>>>>_gfortran_ftell_i2_sub (libgfortran.so.3.0.0)
                                                                      >>>>>>>>_gfortran_ftell_i2_sub (libgfortran.so.3.0.0)
                                                                       ....
                                                                          >>>>>>>>>>>>>_gfortran_st_read (libgfortran.so.3.0.0)
   17.902981000        96.220812461                1 >>>>>>>>>>>>>>__libc_read (libpthread-2.10.2.so)
```

Open | SpeedShop™

Section 4
How to Detect Bottlenecks in Parallel Codes?

Tutorial @ NASA
October 5, 2011

How to use Open|SpeedShop to Analyze the Performance of Parallel Codes.

Integration with MPI

- **O|SS has been tested with a variety of MPIs**
  - Including: Open MPI, MVAPICH[2], MPICH2 (Intel, Cray), and SGI-MPI (MPT)
- **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:
    - **ossmpi** "mpirun –np 128 sweep3d.mpi"
    - **osspcsamp** "mpirun –np 32 sweep3d.mpi"
    - **ossio** "srun –N 4 –n 16 sweep3d.mpi"
    - OFFLINE FULL SYNTAX:
      - **openss –offline –f** "mpirun –np 128 sweep3d.mpi" **hwctime**
    - ONLINE FULL SYNTAX:
      - **openss –online –f** "srun –N 8 –n 128 sweep3d.mpi" **usertime**

Parallel Result Analysis

- **Default views**
  - Values aggregated across all ranks
  - Manually include/exclude individual ranks/processes/threads
- **Comparing ranks**
  - Use the Customize Stats Panel View
  - Create compare columns for process groups or individual ranks
- **Load Balance View**
  - Calculates min, max, average across ranks, processes, or threads
- **Comparative Analysis View (finding outliers)**
  - Uses “cluster analysis” algorithm to group similar performing ranks, processes, or threads

Running MPI Specific Experiments

Offline mpi/mpit/mpiotf experiment

Convenience script basic syntax:

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

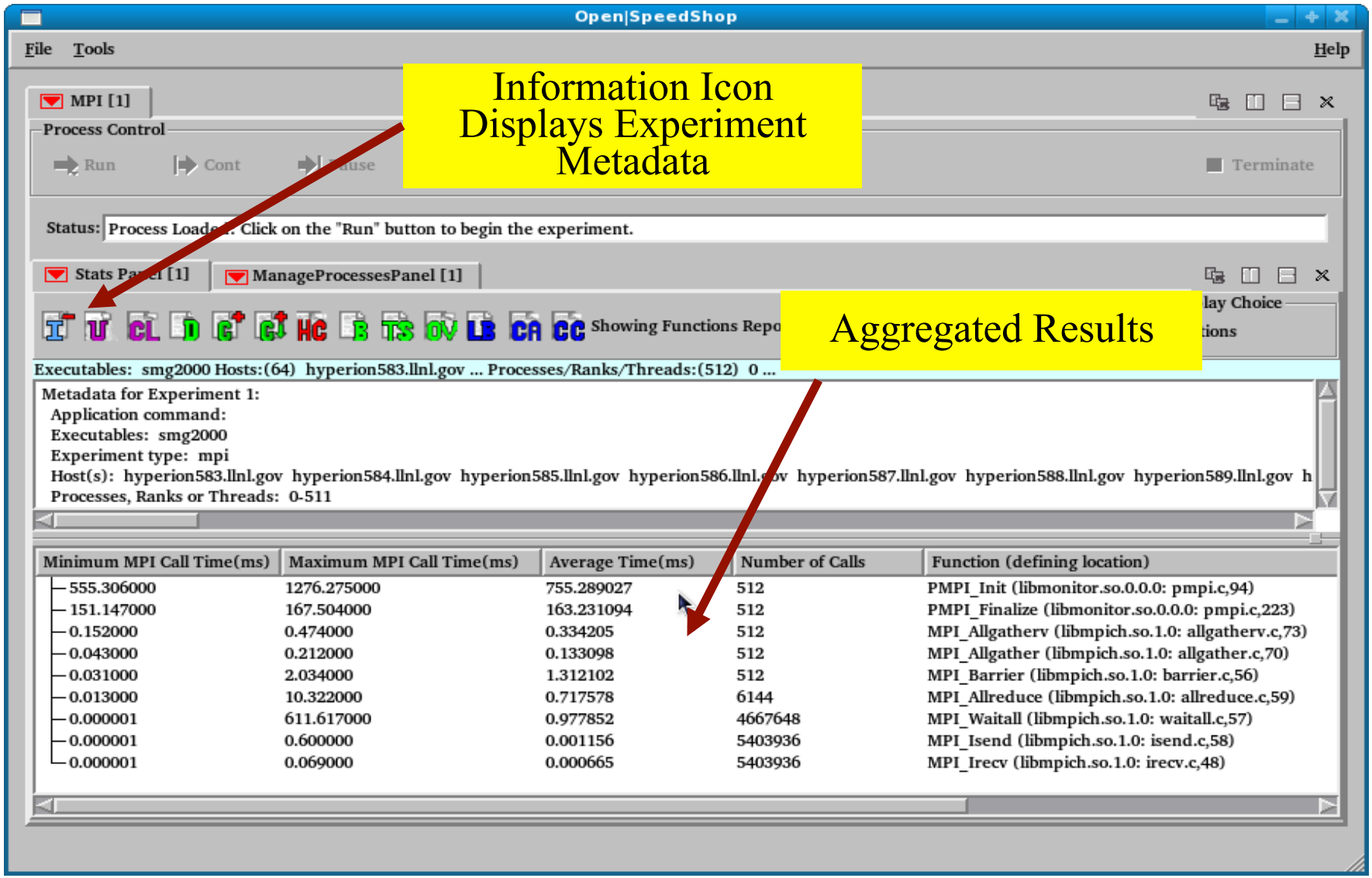
- Parameters
  - Default is all MPI Functions Open|SpeedShop traces
  - MPI Function list to trace (comma separated)
    - MPI_Send, MPI_Recv, ….
  - mpi_category:
    - "all", "asynchronous_p2p", "collective_com", "datatypes", "environment", "graphs_contexts_comms", "persistent_com", "process_topologies", "synchronous_p2p"

Examples:

ossmpi "srun –N 4 –n 32 smg2000 –n 50 50 50"

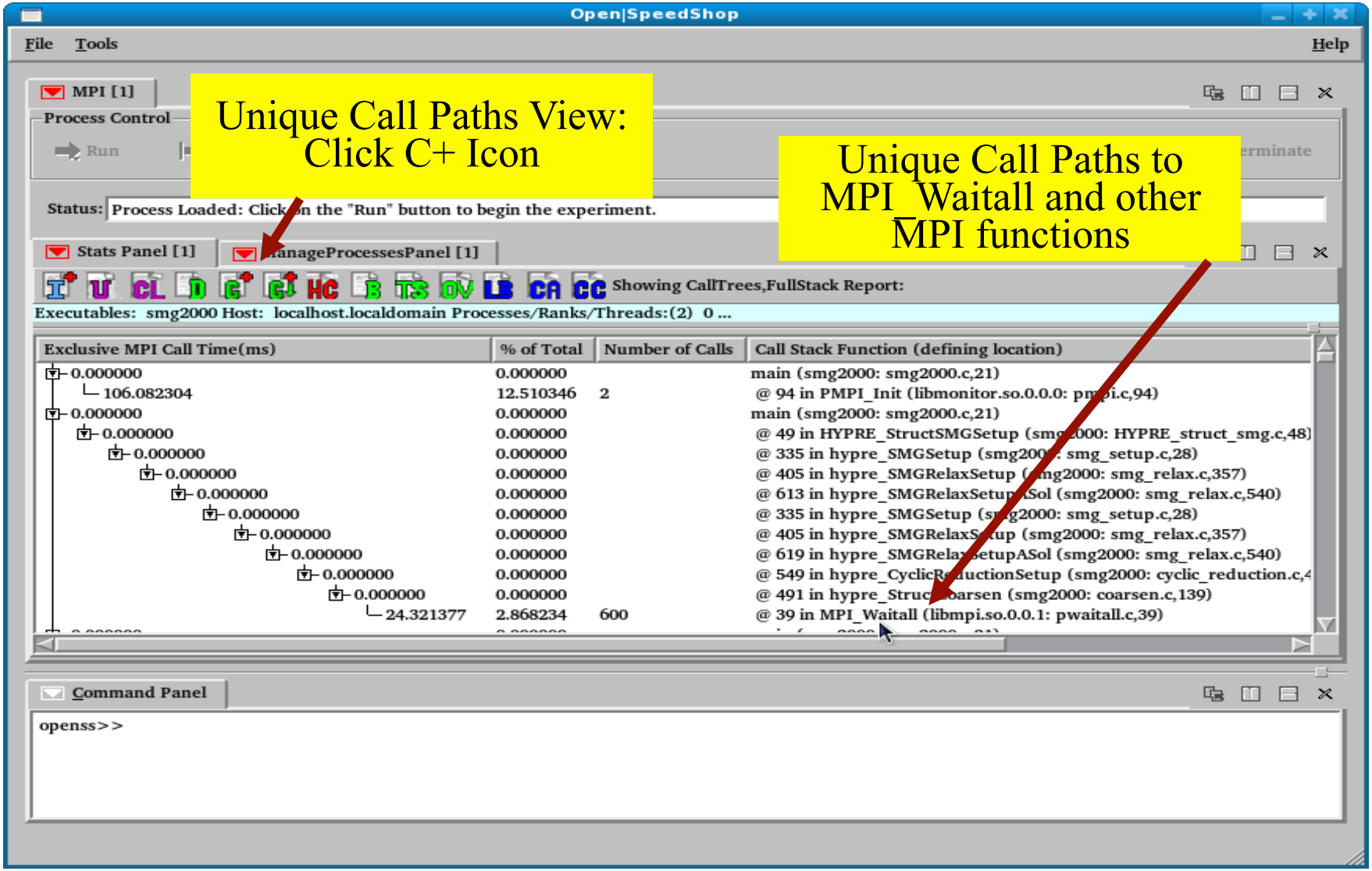
ossmpi "mpirun –np 4000 nbody" MPI_Send,MPI_Recv

MPI Tracing Results: Default View



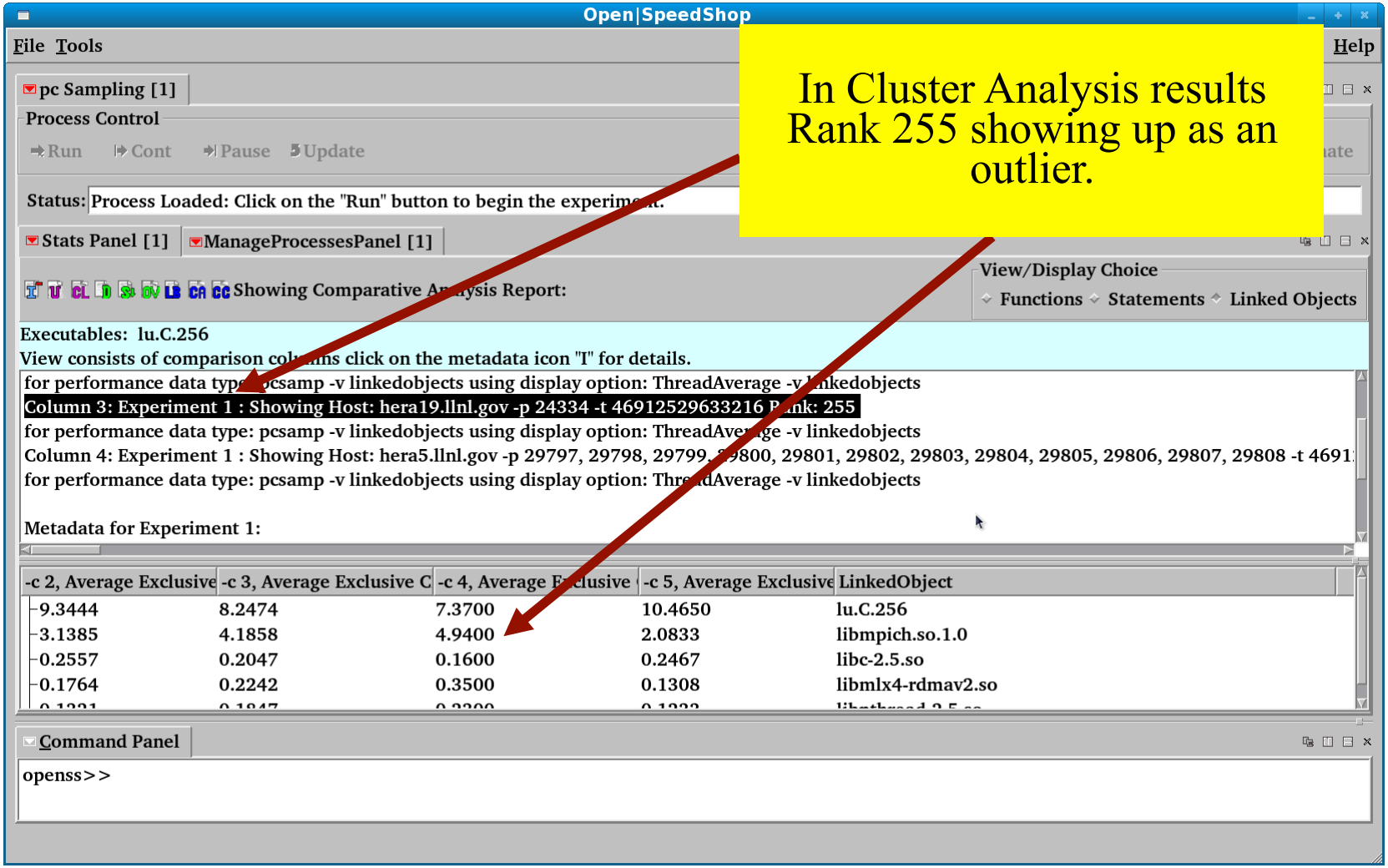
Minimum MPI Call Time(ms)	Maximum MPI Call Time(ms)	Average Time(ms)	Number of Calls	Function (defining location)
555.306000	1276.275000	755.289027	512	PMPI_Init (libmonitor.so.0.0.0: pmpi.c,94)
151.147000	167.504000	163.231094	512	PMPI_Finalize (libmonitor.so.0.0.0: pmpi.c,223)
0.152000	0.474000	0.334205	512	MPI_Allgatherv (libmpich.so.1.0: allgatherv.c,73)
0.043000	0.212000	0.133098	512	MPI_Allgather (libmpich.so.1.0: allgather.c,70)
0.031000	2.034000	1.312102	512	MPI_Barrier (libmpich.so.1.0: barrier.c,56)
0.013000	10.322000	0.717578	6144	MPI_Allreduce (libmpich.so.1.0: allreduce.c,59)
0.000001	611.617000	0.977852	4667648	MPI_Waitall (libmpich.so.1.0: waitall.c,57)
0.000001	0.600000	0.001156	5403936	MPI_Isend (libmpich.so.1.0: isend.c,58)
0.000001	0.069000	0.000665	5403936	MPI_Irecv (libmpich.so.1.0: irecv.c,48)

View Results: Show MPI Callstacks



Finding Outliers: Using NPB: LU

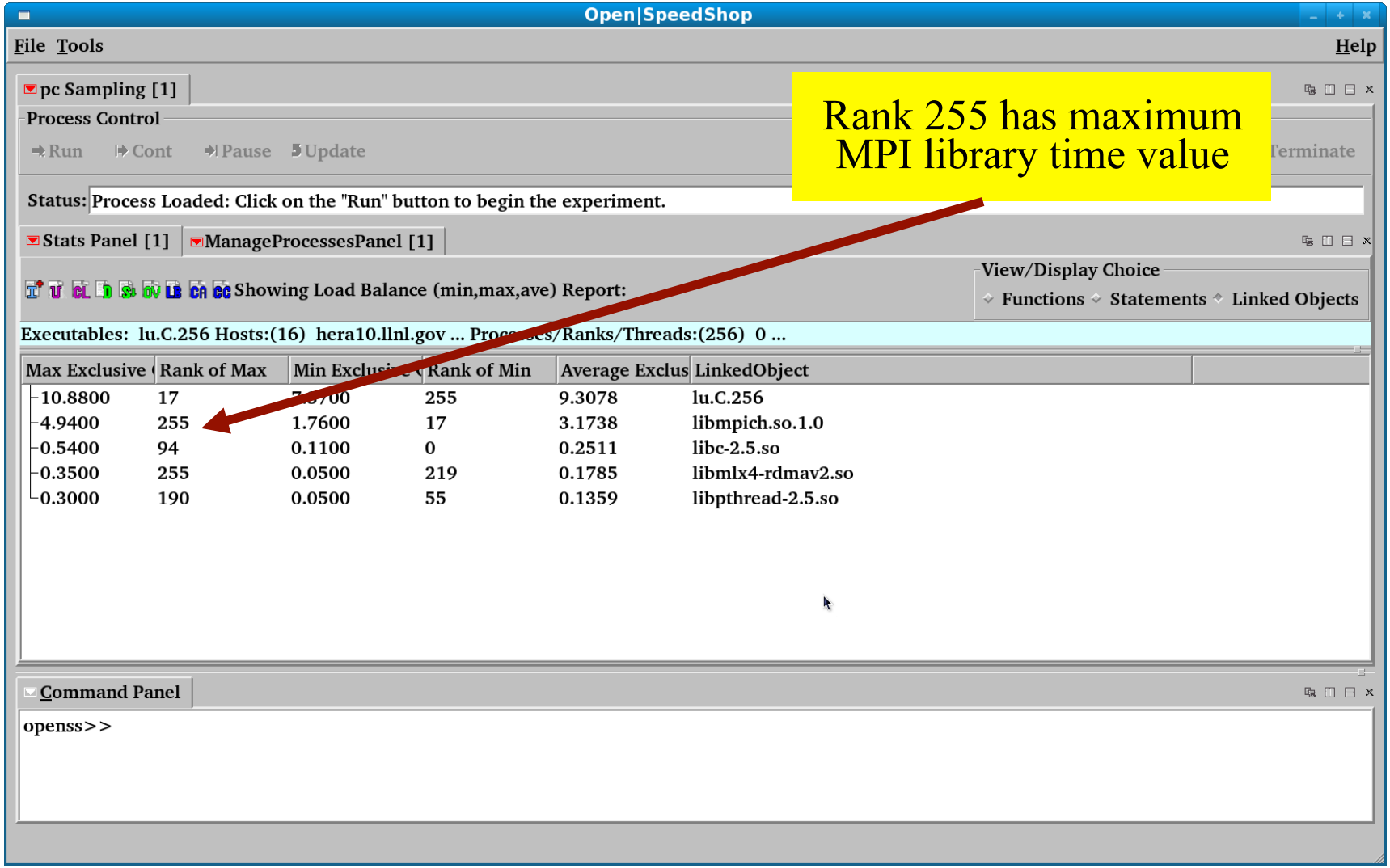
- **Cluster Analysis View based on Linked Objects (libraries)**



In Cluster Analysis results Rank 255 showing up as an outlier.

Load Balance View: Using NPB: LU

- **Load Balance View based on Linked Objects (libraries)**



Open | SpeedShop™

Section 5
User Interfaces

Tutorial @ NASA

October 5, 2011

How to use Open|SpeedShop to Analyze the Performance of Parallel Codes.

CLI Command Overview

- **Experiment creations**
  - expcreate
  - expattach*
- **Experiment control**
  - expgo
  - expwait*
  - expdisable*
  - expenable*
- **Experiment storage**
  - expsave
  - exprestore
- **Result presentation**
  - expview
  - opengui
- **Misc. commands**
  - help
  - list
  - log
  - record
  - playback
  - history
  - quit

*online version only

CLI Command Examples

- **Simple usage to create, run, view data**
  - **openss –cli** (use cli to run experiment:3 commands)
    - **expcreate –f "mutatee 2000"** pcsamp (create an experiment with instrumentation added for the particular collector)
    - **expgo** (runs the experiment gathering data into database)
    - **expview** (displays the default view of the performance data)
- **Alternative views of the performance data**
  - **openss –cli –f <database file name>**
    - **expview –v statements** (see the statements that took the most time)
    - **expview –v linkedobjects** (see time attributed to the libraries in appl.)
    - **expview –v calltrees, fullstack** (see all the call paths in application)
    - **expview –m loadbalance** (see the min, max, average across ranks/ threads)
    - **list –v metrics** (display the optional performance data metrics)
    - **expview –m <metric from above>** (view the metric specified)

Open | SpeedShop™

Section 6
Comparing Experiment Results

Tutorial @ NASA
October 5, 2011

How to use Open|SpeedShop to Analyze the Performance of Parallel Codes.

Comparing Performance Data (1)

- **GUI Custom Compare Panel (CC icon)**
  - See users guide or tutorials for details
- **Convenience Script: osscompare**
  - Compares Open|SpeedShop databases to each other
  - Compare up to 8 at one time
    - If output file name is too long use:
    - export **OPENSS_USE_INTERNAL_NAMING**=1
    - Causes shortened output file names to be created
  - Produces side-by-side comparison listing
  - Optionally create "csv" output for input into spreadsheet (Excel,..)
    - export **OPENSS_CREATE_CSV**=1
- **osscompare syntax:**
  - osscompare "db1.openss, db2.openss" [time, percent, <metric name>] [rows=nn]
    - Where <metric name> means metrics that the cli accepts: "–m <metric name>". This means you can compare on a variety of items.

Comparing Performance Data (2)

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

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

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

-c 2, Exclusive CPU time in seconds. | -c 4, Exclusive CPU time in seconds. | Function (defining location)

3.870000000	3.630000000	hypre_SMGResidual (smg2000: smg_residual.c,152)
2.610000000	2.860000000	hypre_CyclicReduction (smg2000: cyclic_reduction.c,757)
2.030000000	0.150000000	opal_progress (libopen-pal.so.0.0.0)
1.330000000	0.100000000	mca_btl_sm_component_progress (libmpi.so.0.0.2: topo_unity_component.c,0)
0.280000000	0.210000000	hypre_SemiInterp (smg2000: semi_interp.c,126)
0.280000000	0.040000000	mca_pml_ob1_progress (libmpi.so.0.0.2: topo_unity_component.c, 0)

Open | SpeedShop™

Section 7
Static Binary Support

Tutorial @ NASA
October 5, 2011

How to use Open|SpeedShop to Analyze the Performance of Parallel Codes.

Workflow Differences for Static Binary

- **When shared library support is limited**
  - Normal manner of running experiments doesn't work
  - Need to link our collectors into the static executable
- **Current method (static) of support on Cray and BG/P**
- **Must relink application to add in OSS collector code.**
  - **osslink** command is provided to help
- **Have the Open|SpeedShop target NOT host environment module loaded when relinking**

Open|SpeedShop & Static Executables

- **osslink: A script to help with linking in our collectors**
  - **osslink** is a script that hides a lot of the link details
  - Calls to it are usually embedded inside application makefiles
  - Can also be used to compile and link applications
  - Sorts the experiment specific library and collector specification
  - Sorts out some platform differences to do the correct link
- **Find Makefile target to add entry for relink with collector**
  - The user generally needs find the target that creates the actual static executable and create a collector target that links in the selected collector as shown in the example.
- **Example follows ->**

Re-linking application using osslink

- **Example modification for smg2000 application**

```
smg2000: smg2000.o
    @echo  "Linking" $@ "... "
    ${CC} -o smg2000 smg2000.o ${LFLAGS}

 smg2000-pcsamp: smg2000.o
    @echo  "Linking" $@ "... "
    osslink  -v -c pcsamp ${CC} -o smg2000-pcsamp smg2000.o ${LFLAGS}

 smg2000-usertime: smg2000.o
    @echo  "Linking" $@ "... "
    osslink -v -c usertime ${CC} -o smg2000-usertime smg2000.o ${LFLAGS}

 smg2000-hwcsamp: smg2000.o
    @echo  "Linking" $@ "... "
    osslink -v -c hwcsamp ${CC} -o smg2000-hwcsamp smg2000.o ${LFLAGS}

 smg2000-io: smg2000.o
    @echo  "Linking" $@ "... "
    osslink -u open -v -c io ${CC} -o smg2000-io smg2000.o ${LFLAGS}

 smg2000-iot: smg2000.o
    @echo  "Linking" $@ "... "
    osslink -u open -v -c iot ${CC} -o smg2000-iot smg2000.o ${LFLAGS}

 smg2000-mpi: smg2000.o
    @echo  "Linking" $@ "... "
    osslink -v -c mpi ${CC} -o smg2000-mpi smg2000.o ${LFLAGS}
```

Workflow Differences for Static Binary

- **Running the re-linked executable causes the application to write the raw data files to the location specified by OPENSS_RAWDATA_DIR**
- **Converting raw data to an OpenSpeedShop database**
  - Normally the convenience scripts or **openss** tool will do this under the hood
  - ossutil command will create the database file.
  - If raw data files are in /home/jgalaro/smg2000/test/raw
  - **Then ossutil** /home/jgalaro/smg2000/test/raw # creates a database file
  - Symbol table information is read and mapped to the raw data application addresses to map source to the performance data.
- The **ossutil** step may be added to the batch script to eliminate an additional step
- Now normal Open|SpeedShop GUI and CLI viewing of the performance data is possible

Workflow Differences for Static Binary

❖ Now run executable: (note –bb argument to aprun)

PBS -q debug
#PBS -N smg2000-pcsamp
…

 # must have a clean raw data directory each run

rm -rf /home/jgalaro/smg2000/test/raw
mkdir /home/jgalaro/smg2000/test/raw

setenv OPENSS_RAWDATA_DIR /home/jgalaro/smg2000/test/raw
setenv OPENSS_DB_DIR /home/jgalaro/smg2000/test/

cd /home/jgalaro/smg2000/test

 # needs –bb to have the original executable available when doing ossutil
aprun -bb -n 16 /home/jgalaro/smg2000/test/smg2000-pcsamp

 # creates a X.0.openss database file, may need to load module pointing to openspeedshop here

ossutil /home/jgalaro/smg2000/test/raw

Open|SpeedShop on Cray/BG Systems

- **Changes to shared library support "recently"**
  - Dynamic shared library support now available in newer O/S
  - We are working on providing shared support in addition to static binary support
  - This will allow the normal Open|SpeedShop workflow to be used, for example:
    - osspcsamp "how you run the executable dynamically"
- **Have this shared support in Open|SpeedShop for the Cray platform**
- **Need to do additional work to support this on the Blue Gene.**
  - Add code to read the address information for DSOs, Application from alternative location on BG/P.

Open | SpeedShop™

Section 8
Analysis of parallel codes: Conditional Gathering

Tutorial @ NASA
October 5, 2011

How to use Open|SpeedShop to Analyze the Performance of Parallel Codes.

Conditional Gathering of Data (MPI)

MPI_Pcontrol Usage in Open|SpeedShop

This feature allows the user to gather data for sections of their code bounded by **MPI_Pcontrol** calls inserted into the user application. Examples of the execution using this feature need to follow this pattern:

- Modification of application source code to insert MPI_Pcontrol (argument) calls.
  - **MPI_Pcontrol**(1) enables gathering of performance data.
  - **MPI_Pcontrol**(0) disables gathering of performance data.

Conditional Gathering of Data (MPI)

- Set environment variable: **OPENSS_ENABLE_MPI_PCONTROL**=1 to activate the **MPI_Pcontrol** call recognition, otherwise **MPI_Pcontrol** calls will be ignored

- Optionally, set **OPENSS_START_ENABLED**=1 to have performance data gathered until a **MPI_Pcontrol** (0) call is encountered. Otherwise, **NO** performance data will be gathered until a **MPI_Pcontrol** (1) call is encountered. NOTE: **OPENSS_ENABLE_MPI_PCONTROL** must be set for this environment variable to have any effect.

Open | SpeedShop™

Section 9
Other Projects Related to Open|SpeedShop

Tutorial @ NASA

October 5, 2011

How to use Open|SpeedShop to Analyze the Performance of Parallel Codes.

Other work related to OpenSpeedShop

- **Component Based Tool Framework (CBTF) project**
  - Provides the ability to:
    - Create “black box” components
    - Connect these components into a component network
    - Distribute these components and/or component networks across a tree base transport mechanism (MRNet).
    - Want these components to be generic enough to be reused
    - Aimed at giving the ability to rapidly prototype performance tools
    - Create specialized tools to meet the needs of application developers
  - Goal is to recreate OpenSpeedShop from the reusable components

- **NASA SBIR phase I award**
  - We completed a feasibility study & prototyping a new OpenSpeedShop GUI in hopes of winning a phase II award

Open | SpeedShop™

Conclusions

Tutorial @ NASA
October 5, 2011

How to use Open|SpeedShop to Analyze the Performance of Parallel Codes.

Questions vs. Experiments

- **Where do I spend my time?**
  - Flat profiles (pcsamp)
  - Getting inclusive/exclusive timings with callstacks (usertime)
  - Identifying hot callpaths (usertime + HC analysis)
- **How do I analyze cache performance?**
  - Measure memory performance using hardware counters (hwc)
  - Compare to flat profiles (custom comparison)
  - Compare multiple hardware counters (N x hwc, hwcsamp)
- **How to identify I/O problems?**
  - Study time spent in I/O routines (io)
  - Compare runs under different scenarios (custom comparisons)
- **How do I find parallel inefficiencies?**
  - Study time spent in MPI routines (mpi)
  - Look for load imbalance (LB view) and outliers (CA view)

Post-Installation Setup (1)

- **Important runtime environment variables**
  - **OPENSS_PREFIX**
    - Install directory path (if used to set the other environment variables)
  - **OPENSS_PLUGIN_PATH**
    - Path to directory where plugins are stored
  - **OPENSS_MPI_IMPLEMENATION**
    - If Open|SpeedShop was built with multiple MPI implementations, this points openss at the one you are using in your application
    - Also, only required if using the mpi, mpit, or mpiotf experiments
  - **LD_LIBRARY_PATH, PATH**
    - Linux path variables that are typically set to find binaries and libraries

Post-Installation Setup (2)

- **Important runtime environment variables**
  - **OPENSS_RAWDATA_DIR**
    - Shared (across nodes) file system directory path
      - Where raw data files are written
  - **OPENSS_DB_DIR**
    - File system with locking enabled: directory path for database
    - *lustre* file systems sometimes do not have locking enabled
  - Example:

```
export OPENSS_PREFIX=/home/skg/local
export OPENSS_MPI_IMPLEMENTATION=openmpi
export OPENSS_PLUGIN_PATH=$OPENSS_PREFIX/lib64/openspeedshop
export OPENSS_RAWDATA_DIR=/opt/shared
export LD_LIBRARY_PATH=$OPENSS_PREFIX/lib64:$LD_LIBRARY_PATH
export PATH=$OPENSS_PREFIX/bin:$PATH
```

NASA Specific Information

Open|SpeedShop Availability on NASA Systems:

FOR PLEIADES (pfe):

module load /u/jgalarow/privatemodules/openss-201b8-mvapich2
module load /u/jgalarow/privatemodules/openss-201b8-mpt
module load /u/jgalarow/privatemodules/openss-201b8-intel-mpi

FOR COLUMBIA (cfe2):

module load /u/jgalarow/privatemodules/openss-201b6-mpt
module load /u/jgalarow/privatemodules/openss-201b6-intel-mpi

NOTE: If you are using one of the mpi experiments (mpi, mpit, mpiotf) then use the module file that corresponds to the mpi implementation used to create you application. Otherwise, it does not matter which module file is loaded.

Open|SpeedShop Documentation

- **Open|SpeedShop User Guide Documentation**
  - http://www.openspeedshop.org/docs/user_guide/
  - …/share/doc/packages/OpenSpeedShop/users_guide
- **Python Scripting API Documentation**
  - http://www.openspeedshop.org/docs/pyscripting_doc/
  - …/share/doc/packages/OpenSpeedShop/pyscripting_doc
- **Command Line Interface Documentation**
  - http://www.openspeedshop.org/docs/user_guide/
  - …/share/doc/packages/OpenSpeedShop/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

Availability and Contact

- **Current version: 2.0.1 about to be released**
- **Open|SpeedShop Website**
  - http://www.openspeedshop.org/
- **Open|SpeedShop Forum**
  - http://www.openspeedshop.org/forums/
- **Download options:**
  - Package with install script
  - Source for tool and base libraries
- **Feedback**
  - Bug tracking available from website
  - Contact information on website
  - Feel free to contact presenters directly