2016

NASA Open|SpeedShop Tutorial Exercises

HANDS-ON EXERCISES

DECEMBER, 12, 2016

Table of Contents

OpenSpeedShop Tutorial Hands-on EXERCISE overview

This document outlines exercises that are meant to allow users to familiarize themselves with the capabilities in the Open|SpeedShop application performance tool are with respect to analyzing the performance of applications to identify and locate a number of key performance bottleneck types. These are the bottlenecks covered by these tutorial exercises:

- Where my application is taking the most time?
- What are the call paths and function call relationships?
- What are the performance counter values indicating about my application in relationship to the hardware?
- Am I using I/O properly in my application?
- Are there threading or OpenMP usage issues in my application?
- Am I using MPI properly in my application?
- Are there memory usage issues in my application? Am I freeing all the memory that was allocated? What is the high-water mark?
- Are there bottlenecks in my CUDA application? Is the balance of CPU/GPU tuned? Am I spending more time transferring data than time spent in the GPU?

OpenSpeedShop Tutorial Hands-on EXERCISE list

There have a number of performance analysis exercises for the applications in the OpenSpeedShop tutorial exercises directory.

There is a README file for each exercise located in each of the specific exercise directories.

Here is a list of the applications and exercises that are available and their location:

- **Where my application is taking the most time?**
  - Program Counter Sampling, first experiment
    - Section 2: sequential loop_check
    - Found in $HOME/exercises/loop_check

- **What are the call paths and function call relationships?**
  - Call path profiling, Comparisons
    - Section 3: seq_lulesh
    - Found in $HOME/exercises/seq_lulesh

- **Are there threading or OpenMP usage issues in my application?**
  - Parallel/OpenMP
    - Section 4) matmul
    - Found in $HOME/exercises/matmul

- **Am I using MPI properly in my application?**
  - Parallel/MPI
    - Section 4) mpi nbody
    - Found in $HOME/exercises/mpi_nbody

- **What are the performance counter values indicating about my application in relationship to the hardware?**
  - Hardware Counters
    - Section 5) soa_aos
    - Found in $HOME/exercises/soa_aos

- **Am I using I/O properly in my application?**
  - Input/Output
    - Section 6: IOR
    - Found in $HOME/exercises/IOR

- **Are there memory usage issues in my application? Am I freeing all the memory that was allocated? What is the high-water mark?**
  - Memory Analysis
    - Section 7: matmul
    - Found in $HOME/exercises/matmul

- **Are there bottlenecks in my CUDA application? Is the balance of CPU/GPU tuned? Am I spending more time transferring data than time spent in the GPU?**

- NVIDIA Cuda Analysis
  - Section 8: GEMM, FFT
  - Found in $HOME/exercises/cuda/shoc/bindir/bin/EP/CUDA

OpenSpeedShop Tutorial Hands-on EXERCISE-1

This directory contains a simple C program containing three compute loops. One loop does 5000 iterations, another does 10000, and the last loop does 15000 iterations.

With the pcsamp experiment we should see proportional percentage for each loop.

Exercise 1 Assignment:

1) Build the code by typing: 'make'
2) Run the executable 'loop_check' with: ./loop_check
   - Use the run_alone_qsub.sh script using
     - qsub run_alone_qsub.sh
4) Analyze performance of the code with OpenSpeedShop experiment 'pcsamp'.
   - Run by using the osspcsamp convenience script:
     - osspcsamp "./loop_check"
   - Use the run_pcsamp_qsub.sh script to accomplish the task
     - qsub run_pcsamp_qsub.sh
5) Are the percentages for each of the loops proportional to 5/30, 10/30, and 15/30?
   - Sampling has statistical variations preventing precise match.
   - You may look at the OpenSpeedShop run output or use the GUI:
     - openss -f loop_check-pcsamp.openss
   - or you may use the CLI:
     - openss -cli -f loop_check-pcsamp.openss
   - then use "expview" command in some of its common forms:
     - expview
     - expview -v statements
     - expview -v loops
     - expview -v linkedobjects

OpenSpeedShop Tutorial Hands-on EXERCISE-2

This is the sequential application, lulesh, based OpenSpeedShop performance analysis exercise.

Exercise 2 Assignment:

1) Compile the lulesh application
   Execute these commands:
   make clean; make

2) Run the usertime experiment on the lulesh application.

   Use:
   qsub run_usertime_qsub.sh
   to run the usertime experiment.

   Results returned:
   An Open|SpeedShop database file with the call path profiling performance information and the lulesh symbol information.

3) Examine the performance information gathered by Open|SpeedShop.

- To open the database file with the GUI:
  - openss -f <database file name>
  - most likely: openss -f lulesh-usertime-0.openss
- As more usertime experiments are run, OpenSpeedShop increments to: lulesh-usertime-n.openss
- Identify the function that took the most time.
  - Is it: CalcHourglassControlForElems ?
- Find which six functions that are called by main
  - Hint look for the Butterfly view icon.
- Find the hot call paths in this program
  - Hint look for the Hot Call path view icon
  - Does one call path take much longer than the rest? It is a short one.
- Find the all the call paths in this program that contain the function CalcKinematicsForElems
  - Hint look for the C icon the red + and green downward arrow.
  - Select the CalcKinematicsForElems from the default function list and then click the icon. This will list all the call paths that the selected function is included.
- Identify the statement that took the most time.
  - For this one must select Statements as the view granularity (View/Display Choice)
  - Then click on the Default View icon (D).

    - The statement that took the most time is: line 889 in lulesh.cc. The "(889)" indicates the line number.

- Display the statement that took the most time in the GUI.
    - From the previous step, the lulesh.cc (889) line of data is displayed.
    - Double click that line to display the source

- Identify the loop that took the most time.
    - For this one must select Loops as the view granularity (View/Display Choice)
    - Then click on the Default View icon (D).

- Display the loop that took the most time in the GUI.
    - From the previous step, the lulesh.cc (480) line of data is displayed.
    - Double click that line to display the source

4) Advanced: CLI exercises: (please use the Quick Start Guide for help and/or see the hints file)

- Open the database file with the CLI:
    - openss -cli -f <database file name>
    - Most likely named: openss -cli -f lulesh-usertime-0.openss

- List the default view (functions)
    - Hint: expview

- List the default view (statements)
    - Hint: expview -v statements

- List the default view (loops)
    - Hint: expview -v loops

- List the top five (5) call paths that took the most time using the CLI.
    - Note: same output as the (HC icon) Hot Call path view provides.
    - Hint: expview -vfullstack usertime5

- List the functions that call the top time taking function in the application run
    - This is the CLI version of the GUI (B icon) butterfly view in the GUI.
    - Hint: expview -vbutterfly -f CalcHourglassControlForElems

5) Advanced: Comparison of databases to detect changes in performance for the application being run

- mv lulesh-usertime-0.openss lulesh-usertime-O3.openss
- vi Makefile $ change -O3 to -O1

- make clean ; make
- qsub run_usertime_qsub.sh
- mv lulesh-usertime-0.openss lulesh-usertime-O1.openss
- Compare -O1 compiled version of lulesh to the -O3 version
- Compare functions
    - osscompare "lulesh-usertime-O1.openss,lulesh-usertime-O3.openss"
- Compare statements
    - osscompare "lulesh-usertime-O1.openss,lulesh-usertime-O3.openss" viewtype=statements
- Note how the function and statement results improved with using -O3

OpenSpeedShop Tutorial Hands-on EXERCISE-3

The purpose of this exercise is to become familiar with performance analysis of MPI applications both with sampling experiments and with the MPI specific experiments (mpip, mpi, mpit).

- mpip
  - A lightweight profiling of the MPI function calls
- mpi
  - A tracing of MPI function calls, recording call paths, number of times called, time spent in the MPI functions.
- mpit
  - A tracing of MPI function calls, recording call paths, number of times called, time spent in the MPI functions, and the arguments and values in each call.
  - A chronological list of the MPI function calls and arguments is also available (-vtrace in the CLI, EL icon in GUI).

Exercise 3 Assignment:

1) Build nbody application
   make clean; make

2) Run the nbody application with the listed experiments

2a) pcsamp (program counter sampling which gives an overview of where time is spent)
   qsub run_pcsamp_qsub.sh

2b) mpip (lightweight MPI profiling which doesn't save individual calls for a chronological list)
   qsub run_mpip_qsub.sh

2c) mpit (MPI tracing which does include argument values and saves information about each MPI function call)
   qsub run_mpit_qsub.sh

3) Examine the performance information that was gathered by Open|SpeedShop for the pcsamp experiment

- To open the database file with the GUI:
  - openss -f <database file name>
  - most likely: openss -f nbody-pcsamp-0.openss

- Identify the function that took the most time.
  - The default view sorts by most time taken and it displays functions by default.
  - So, the function that took the most time is: main

- Identify the statement that took the most time.

    - For this one must select Statements as the view granularity (View/Display Choice)
        - and then click on the Default View icon (D).
    - With regards to the statement that took the most time is: line 242 in nbody-mpi.c, the "(242)" is the line number.

- Display the statement that took the most time in the GUI.
    - From the previous step, the nbody-mpi.c (242) line of data is displayed.
    - Double click that line to display the source

- Determine if this run of the application has balance across all ranks (no load imbalance)
    - Select the view granularity (View/Display Choice). Functions or LinkedObjects are good choices.
    - Then choose the Load Balance (LB) view to display the minimum, maximum, and average time display.
    - Look at the minimum, maximum, and average values across the functions. Are they relatively close in values?
    - If they are not, you can select the Cluster Analysis (CA) view, which groups like performing ranks into groups.
    - If there are outlying functions, then there will be multiple columns of data in the Cluster Analysis view.
    - To identify which ranks, threads, or processes are in each group, click the Information icon (I+) to see the
    - metadata for each group.

4) Examine the performance information that was gathered by Open|SpeedShop for the mpip experiment

- To open the database file with the GUI:
    - openss -f <database file name>
    - openss -f nbody-mpip-0.openss

- Are the min, max, and average values, per call for all the MPI functions relatively the same?
    - If they are then, we can assume that the application if balanced across all ranks
    - This is expected for this small run, but in the real world, there may be differences

- Let us assume that the load balance shown in the default view was not well balanced.
    - The next step would be to click on the CA icon which does a cluster analysis algorithm that would display multiple columns in the StatsPanel.
    - Each column would represent a rank or set of ranks that are like performing. This is one way to find which ranks are the outliers.

- The HC icon (hot call path) can call paths taking a long time. Because this application isn't doing a lot of work (because need

smaller time runs for tutorial), the paths are likely to MPI_Init, which is not interesting. But be on the lookout for paths to MPI_Waitall, MPI_Allgather, etc. that show wait times that cause imbalance.

5) Examine the performance information that was gathered by Open|SpeedShop for the mpit experiment

- To open the database file with the GUI:
  - openss -f <database file name>
  - openss -f nbody-mpit-0.openss

- Are the min, max, and average values per rank for all the MPI functions relatively the same?
  - If they are then, we can assume that the application if balanced across all ranks
  - This is expected for this small run, but in the real world, there may be differences

- Let us assume that the load balance shown in the default view was not well balanced.
  - The next step would be to click on the CA icon which does a cluster analysis algorithm that would display multiple columns in the StatsPanel.
  - Each column would represent a rank or set of ranks that are like performing. This is one way to find which ranks are the outliers.

- The EL icon will produce a chronological list of all the MPI function calls and the arguments to each call.
  - This can be an extremely long list in real life programs.
  - Some users of O|SS use the CLI via the "expview -vtrace > save_to_file.txt" to save the data and mine it out of the O|SS tools for patterns of MPI calls that might be problematic.
  - In this example the number of calls is reasonable, so try the EL icon to see the information for each call.

- Note the size of the mpit database compared to the mpip database. If one can examine the load balance with the mpip experiment, that would be the best route to go with. mpit databases get very large on long running programs.

OpenSpeedShop Tutorial Hands-on EXERCISE-4

This is an OpenMP application performance analysis task. We examine the wait times associated with running an OpenMP application, matmul.

1) Please do these "make clean; make" commands in this directory to build matmul
   - make clean; make

2) Run two experiments: osspcsamp and ossomptp (OpenMP specific) by:
   - qsub run_parallel_qsub.sh
   - qstat -u <login id> will monitor the job status

3) Examine the performance information in the pcsamp database file

   - To open in the GUI:
     - openss -f matmul-pcsamp-0.openss

4) Examine the performance information in the omptp database file
   - To open in the CLI:
     - openss –cli -f matmul-omptp-0.openss

Exercise 4: Context, Explanation, and Examples of the OMPTP experiment.

Here we use a very simple OpenMP matrix multiply example to demonstrate how omptp is currently profiling an OpenMP application via the OMPT API.

In this directory is a simple matrix multiply example from the ompt-test-suite.

The omptp collector in use is from the ompt-branch of cbtf-krell.
The omptp collector uses these symbol names to record idle,barrier,wait_barrier times.
These names are not locked in stone and simple IDLE,BARRIER,WAIT_BARRIER could suffice.
The descriptions below are based on the latest ompt working document and describe their meaning in the omptp collector.

OMPT_THREAD_IDLE:
Time spent during the period when a thread starts to idle outside a parallel region and when a thread finishes idling outside a parallel region.

OMPT_THREAD_BARRIER:
Time spent during the period when an implicit task begins execution of a barrier region and after an implicit task exits a barrier region.

OMPT_THREAD_WAIT_BARRIER:

Time spent during the period when an implicit task starts to wait in a barrier region and when an implicit task finishes waiting in a barrier region. One barrier region may generate multiple pairs of barrier begin and end callbacks in a task. e.g. if waiting at the barrier occurs in multiple stages or if another task is scheduled on this thread while it waits at the barrier.

This is the naming for implicit tasks within a parallel region function: _omp_fn.N: The parallel region context or work being parallelized by an implicit task. Time spent during the period when an implicit task is fully initialized but before the task executes its work to after an implicit task executes its closing synchronization barrier but before returning to idle or the task is destroyed.

The compare view below is by function. Note that the initialize function is called three times (once per matrix - count metric not shown here). The OMP_* entries are aggregated across all regions. One can see the individual contribution per region (shown later).

openss>>expcompare -t0:4 -mtime -vsummary

-t 0, Exclusive times in seconds.	-t 2, Exclusive times in seconds.	-t 3, Exclusive times in seconds.	-t 4, Exclusive times in seconds.	Function (defining location)
10.602988	10.602989	10.523003	10.222533	compute._omp_fn.1 (matmult: matmult.c,68)
0.557765	0.548475	0.465930	0.417029	compute_interchange._omp_fn.3 (matmult: matmult.c,118)
0.274503	0.224162	0.164765	0.090968	compute_triangular._omp_fn.2 (matmult: matmult.c,95)
0.106101	0.000087	0.000095	0.000106	OMPT_THREAD_BARRIER (omptp-collector-monitor-mrnet.so: collector.c,596)
0.105928	0.000000	0.000000	0.000000	OMPT_THREAD_WAIT_BARRIER (omptp-collector-monitor-mrnet.so: collector.c,611)
0.002009	0.001309	0.001319	0.001112	initialize._omp_fn.0 (matmult: matmult.c,32)
0.000000	0.010206	0.171964	0.531463	OMPT_THREAD_IDLE (omptp-collector-monitor-mrnet.so: collector.c,582)
11.649295	11.387227	11.327076	11.263210	Report Summary

These are the parallel regions source code locations which the SourcePanel would display.

compute at matmult,68: #pragma omp parallel private(i,j,k) shared(a,b,c)
compute_interchange at matmult,118: #pragma omp parallel private(i,j,k) shared(a,b,c)
compute_triangular at matmult,95: #pragma omp parallel private(i,j,k) shared(a,b,c)
initialize at matmult,32: #pragma omp parallel private(i,j) shared(matrix)

The symbolic information for the OMPT_* names is meaningless in terms of SourcePanel views.

barrier,wait_barrier, and idle are mapped to the specific parallel regions.

Show time for calltrees in thread 2.
compute.omp_fn.1 is implicit task time for thread 2 in this region.
The idle and barrier entries below the region are those times for this region.

```
openss>>expview -vcalltrees -t2 -f compute._omp_fn.1 -m time

Exclusive  Call Stack Function (defining location)
 times in
 seconds.
        __clone (libc-2.18.so)
        >start_thread (libpthread-2.18.so)
        >>__kmp_launch_worker(void*) (libiomp5.so: z_Linux_util.c,701)
        >>>__kmp_launch_thread (libiomp5.so: kmp_runtime.c,5419)
11.741148  >>>>compute._omp_fn.1 (matmultA: matmult.c,68)
 0.204475  >>>>>OMPT_THREAD_IDLE (omptp-collector-monitor-mrnet.so: collector.c,582)
 0.000011  >>>>>OMPT_THREAD_BARRIER (omptp-collector-monitor-mrnet.so: collector.c,596)
```

Show time for all calltrees for barrier and wait_barrier.

```
openss>>expview -v calltrees -f *BARRIER -m time

Exclusive  Call Stack Function (defining location)
 times in
 seconds.
        _start (matmultA)
        >__libc_start_main (libmonitor.so.0.0.0: main.c,541)
        >>__libc_start_main (libc-2.18.so)
        >>>main (matmultA: matmult.c,161)
        >>>>do_work (matmultA: matmult.c,137)
        >>>>>compute (matmultA: matmult.c,63)
        >>>>>>__kmp_api_GOMP_parallel_start_10_alias (libiomp5.so: kmp_gsupport.c,456)
        >>>>>>>compute._omp_fn.1 (matmultA: matmult.c,68)
 0.003809  >>>>>>>>OMPT_THREAD_BARRIER (omptp-collector-monitor-mrnet.so: collector.c,596)
 0.003766  >>>>>>>>OMPT_THREAD_WAIT_BARRIER (omptp-collector-monitor-mrnet.so: collector.c,611)
        >>>>>initialize (matmultA: matmult.c,28)
        >>>>>>__kmp_api_GOMP_parallel_start_10_alias (libiomp5.so: kmp_gsupport.c,456)
        >>>>>>>initialize._omp_fn.0 (matmultA: matmult.c,32)
 0.000336  >>>>>>>>OMPT_THREAD_BARRIER (omptp-collector-monitor-mrnet.so: collector.c,596)
 0.000178  >>>>>>>>OMPT_THREAD_WAIT_BARRIER (omptp-collector-monitor-mrnet.so: collector.c,611)
        >>>>>compute_interchange (matmultA: matmult.c,114)
        >>>>>>__kmp_api_GOMP_parallel_start_10_alias (libiomp5.so: kmp_gsupport.c,456)
        >>>>>>>compute_interchange._omp_fn.3 (matmultA: matmult.c,118)
 0.000148  >>>>>>>>OMPT_THREAD_BARRIER (omptp-collector-monitor-mrnet.so: collector.c,596)
        __clone (libc-2.18.so)
        >start_thread (libpthread-2.18.so)
        >>__kmp_launch_worker(void*) (libiomp5.so: z_Linux_util.c,701)
        >>>__kmp_launch_thread (libiomp5.so: kmp_runtime.c,5419)
        >>>>initialize._omp_fn.0 (matmultA: matmult.c,32)
 0.000134  >>>>>OMPT_THREAD_BARRIER (omptp-collector-monitor-mrnet.so: collector.c,596)
        >__libc_start_main (libmonitor.so.0.0.0: main.c,541)
        >>__libc_start_main (libc-2.18.so)
        >>>main (matmultA: matmult.c,161)
        >>>>do_work (matmultA: matmult.c,137)
        >>>>>compute_interchange (matmultA: matmult.c,114)
        >>>>>>__kmp_api_GOMP_parallel_start_10_alias (libiomp5.so: kmp_gsupport.c,456)
        >>>>>>>compute_interchange._omp_fn.3 (matmultA: matmult.c,118)
```

```
0.000113  >>>>>>>>OMPT_THREAD_WAIT_BARRIER (omptp-collector-monitor-mrnet.so: collector.c,611)
        >>__kmp_launch_worker(void*) (libiomp5.so: z_Linux_util.c,701)
        >>>__kmp_launch_thread (libiomp5.so: kmp_runtime.c,5419)
        >>>>compute._omp_fn.1 (matmultA: matmult.c,68)
0.000047  >>>>>OMPT_THREAD_BARRIER (omptp-collector-monitor-mrnet.so: collector.c,596)
        >>>>compute_interchange._omp_fn.3 (matmultA: matmult.c,118)
0.000038  >>>>>OMPT_THREAD_BARRIER (omptp-collector-monitor-mrnet.so: collector.c,596)
        >>>>compute_triangular._omp_fn.2 (matmultA: matmult.c,95)
0.000030  >>>>>OMPT_THREAD_BARRIER (omptp-collector-monitor-mrnet.so: collector.c,596)
        >>>main (matmultA: matmult.c,161)
        >>>>do_work (matmultA: matmult.c,137)
        >>>>>compute_triangular (matmultA: matmult.c,91)
        >>>>>>__kmp_api_GOMP_parallel_start_10_alias (libiomp5.so: kmp_gsupport.c,456)
        >>>>>>>compute_triangular._omp_fn.2 (matmultA: matmult.c,95)
0.000009  >>>>>>>>OMPT_THREAD_BARRIER (omptp-collector-monitor-mrnet.so: collector.c,596)
```

Show time for all calltrees for idle.

```
openss>>expview -v calltrees -f  *IDLE -m time

Exclusive  Call Stack Function (defining location)
 times in
 seconds.
        __clone (libc-2.18.so)
        >start_thread (libpthread-2.18.so)
        >>__kmp_launch_worker(void*) (libiomp5.so: z_Linux_util.c,701)
        >>>__kmp_launch_thread (libiomp5.so: kmp_runtime.c,5419)
        >>>>compute._omp_fn.1 (matmultA: matmult.c,68)
0.544330  >>>>>OMPT_THREAD_IDLE (omptp-collector-monitor-mrnet.so: collector.c,582)
        >>>>compute_interchange._omp_fn.3 (matmultA: matmult.c,118)
0.001042  >>>>>OMPT_THREAD_IDLE (omptp-collector-monitor-mrnet.so: collector.c,582)
        >>>>initialize._omp_fn.0 (matmultA: matmult.c,32)
0.000763  >>>>>OMPT_THREAD_IDLE (omptp-collector-monitor-mrnet.so: collector.c,582)
```

OpenSpeedShop Tutorial Hands-on EXERCISE-5

Using A OpenSpeedShop Hardware Counter Experiment to aid in Understanding Importance of Data Structures:

The purposes of this exercise are:

- Use a performance analysis tool to understand the importance of data structures in supporting good vectorization and cache utilization through unit stride through the arrays during computation

- A very common computation kernel in scientific computing and visualization is a kernel that computes the distance of a collection of points from a reference point

- In this contrived example to illustrate the importance of data structures:
  - Structure of Arrays (SOA) vs Array of structures (AOS), a simple Fortran serial program is used to count the number of points among a given large collection of points to compute how many of these points lie inside a given sphere in space.

- What does the sample code do? The code aos.F90 arranges the x,y,z coordinates of the points and their distance in space, using array of structures. The same computation with the 3-dimensional points and distance arranged using a structure-of-arrays is in the code soa.F90.

Exercise 5 Assignment:

1) Build the code by typing:
- 'make clean; make' in the directories: soa and aos

2) Run the executable aos.exe and soa.exe in each of the directories: soa and aos via
- qsub run_qsub.sh

3) Observe the run times; How much faster is soa over aos?

4) Analyze performance of the code with OpenSpeedShop experiment 'pcsamp'.
- Run by using the pcsamp convenience script: osspcsamp "./aos.exe" and: osspcsamp "./soa.exe"
- Use the PBS batch script in each directory:
  - qsub run_pcsamp_qsub.sh
- Observe the results using the CLI or GUI:
  - CLI: openss -cli -f aos-pcsamp-0.openss
  - GUI: openss -f aos-pcsamp-0.openss
  - CLI: openss -cli -f soa-pcsamp-0.openss
  - GUI: openss -f soa-pcsamp-0.openss

5) Analyze performance of the code with OpenSpeedShop experiment 'hwcsamp'.

- Run by using the hwcsamp convenience script:
  - osshwcsamp "./**aos**.exe" PAPI_TOT_CYC,PAPI_TOT_INS,PAPI_L1_DCM,PAPI_TLB_DM
  - osshwcsamp "./**soa**.exe" PAPI_TOT_CYC,PAPI_TOT_INS,PAPI_L1_DCM,PAPI_TLB_DM
- Use the PBS batch script in each directory:
  - qsub run_hwcsamp_qsub.sh

6) Observe the differences in L1 cache misses and Total-instructions and Total-cycles and the TLB misses using the CLI or GUI:
- CLI: openss -cli -f aos-hwcsamp-0.openss
- GUI: openss -f aos-hwcsamp-0.openss
- CLI: openss -cli -f soa-hwcsamp-0.openss
- GUI: openss -f soa-hwcsamp-0.openss

- What methods could you use to measure vectorization effectiveness?

7) Run osshwc experiment to pin-point the source lines where the PAPI_L1_DCM (level 1 data cache misses) are occurring.
- Use
  - osshwc "./aos.exe" PAPI_L1_DCM
  - and: osshwc "./soa.exe" PAPI_L1_DCM
- Use the PBS batch script in each directory:
  - qsub run_hwc_qsub.sh

8) Observe the differences in L1 cache misses between the two versions using the CLI or GUI:
- openss -cli -f aos-hwc-0.openss or openss -f aos-hwc-0.openss
- openss -cli -f soa-hwc-0.openss or openss -f soa-hwc-0.openss

8) Compare the databases from each directory:

- Change directories to the top level soa_aos directory.
- Then compare the pcsamp and hwcsamp runs for each aos and soa run to each other using the osscompare script:

- For pcsamp function compare:
  - osscompare "aos/aos.exe-pcsamp-6.openss,soa/soa.exe-pcsamp-0.openss"
- For pcsamp statement compare:
  - osscompare "aos/aos.exe-pcsamp-0.openss,soa/soa.exe-pcsamp-0.openss" viewType=statements

- For hwcsamp function compare:
  - osscompare "aos/aos.exe-hwcsamp-0.openss,soa/soa.exe-hwcsamp-0.openss"
- For hwcsamp statement compare:
  - osscompare "aos/aos.exe-hwcsamp-0.openss,soa/soa.exe-hwcsamp-0.openss" viewType=statements
- For hwcsamp papi_l1_dcm compare:

    - osscompare "aos/aos.exe-hwcsamp-0.openss,soa/soa.exe-hwcsamp-0.openss" papi_l1_dcm
- For hwcsamp papi_tot_cyc compare:
    - osscompare "aos/aos.exe-hwcsamp-0.openss,soa/soa.exe-hwcsamp-0.openss" papi_tot_cyc

- For hwc function compare:
    - osscompare "aos/aos.exe-hwc-0.openss,soa/soa.exe-hwc-0.openss"
- For hwc statement compare:
    - osscompare "aos/aos.exe-hwc-0.openss,soa/soa.exe-hwc-0.openss" viewType=statements

OpenSpeedShop Tutorial Hands-on EXERCISE-6

Using A OpenSpeedShop Input/Output Experiment to aid in Understanding Importance of the Usage of I/O in application development.

Exercise 6 Assignment:

1) Run IOR with OpenSpeedShop

- Run ossiot "mpirun -np 8 --hosts ccn001 ./IOR" by using the PBS batch script:
  - qsub run_iot_qsub.sh
  - qstat -u <login id> # monitor job progress

2) Observe what the maximum and minimum bytes were written during the application run by using the CLI:
- openss -cli -f IOR-iot-0.openss
- HINTS: These values are output in the default view, see 3)

3) Find which call path in the program allocated the most bytes during the application run by using the CLI:
- openss -cli -f IOR-iot-0.openss
- HINTS: The gui doesn't help with this assignment.
- Please open the iot experiment database with:
  - openss -cli -f <database_name>
- openss>>expview <gives default view>

```
openss>>expview -m max_bytes -vfullstack iot4

Max_Bytes  Call Stack Function (defining location)
     Read
  Written
          main (IOR: IOR.c,108)
          > @ 2173 in TestIoSys (IOR: IOR.c,1848)
          >> @ 2611 in WriteOrRead (IOR: IOR.c,2562)
          >>> @ 251 in IOR_Xfer_POSIX (IOR: aiori-POSIX.c,224)
  262144  >>>>__GI___read (libc-2.12.so)
          main (IOR: IOR.c,108)
          > @ 2013 in TestIoSys (IOR: IOR.c,1848)
          >> @ 2608 in WriteOrRead (IOR: IOR.c,2562)
          >>> @ 244 in IOR_Xfer_POSIX (IOR: aiori-POSIX.c,224)
  262144  >>>>write (libc-2.12.so)
```

6) In this application all the ranks are doing the same allocation per rank?

HINT: openss>>expcompare -r0:7 -m max_bytes,tot_bytes

```
      -r 0,        -r 0,      -r 1,        -r 1,  Function (defining
location)
```

```
   Max_Bytes  Total_Bytes  Max_Bytes  Total_Bytes
        Read         Read       Read         Read
     Written      Written    Written      Written
      262144      1048576     262144      1048576  __GI___read (libc-
2.12.so)
      262144      1048576     262144      1048576  write (libc-2.12.so)
```

7) The iot experiment gives a chronological list of I/O events with each function's argument values. One can get this information with the GUI:

- HINT: Hit the EL (event list) icon.
- HINT - CLI: Type expview -v trace

8) Run ossiop "mpirun -np 8 --hosts ccn002 ./IOR"

- Use the PBS batch script:
  - qsub run_iop_qsub.sh
  - qstat -u <login id>   # monitor job progress

9) Observe the differences in the output between the lightweight iop experiment and the heavier weight iot experiment (gathers and keeps more data).

OpenSpeedShop Tutorial Hands-on EXERCISE-7

Using A OpenSpeedShop Memory (mem) Experiment to aid in Understanding Importance of the Usage of memory in application development.

Exercise 7 Assignment:

1) Please do a "make" command in this directory to build matmul
   - make clean; make

2) Run the mem experiment: ossmem by:
   - qsub run_mem_qsub.sh
   - qstat -u <login id> will monitor the job status

3) Analyze the data produced by the ossmem experiment by opening the newly created database with the CLI:
   - openss -cli -f matmul-mem-0.openss

```
$ openss -cli -f matmul-mem-0.openss
openss>>[openss]: The restored experiment identifier is:  -x 1
openss>>expview

Exclusive     % of  Number     Min      Min     Max      Max    Total  Function (defining location)
  (ms)     Total     of  Request  Requested  Request  Requested    Bytes
         Time  Calls   Count    Bytes   Count    Bytes  Requested

 0.024250 88.932082   1546      1      192      6     4096   6320832 __GI___libc_malloc (libc-2.17.so)
 0.002016  7.393282      5                                          __GI___libc_free (libc-2.17.so)
 0.000684  2.508435      7      1      368      1      368      2576 __calloc (libc-2.17.so)
 0.000318  1.166202      1      1       72      1       72        72 __realloc (libc-2.17.so)

openss>>expview -vleaked

Number    Total  Function (defining location)
  of     Bytes
 Leaks   Leaked
  1541  6306752  __GI___libc_malloc (libc-2.17.so)
    1       72  __realloc (libc-2.17.so)

openss>>expview -v unique

Exclusive     % of  Number     Min      Min     Max      Max    Total  Function (defining location)
  (ms)     Total     of  Request  Requested  Request  Requested    Bytes
         Time  Calls   Count    Bytes   Count    Bytes  Requested

 0.024250 88.932082   1546      1      192      6     4096   6320832 __GI___libc_malloc (libc-2.17.so)
 0.002016  7.393282      5                                          __GI___libc_free (libc-2.17.so)
 0.000684  2.508435      7      1      368      1      368      2576 __calloc (libc-2.17.so)
 0.000318  1.166202      1      1       72      1       72        72 __realloc (libc-2.17.so)
```

```
openss>>expview -v highwater

Number     Total  Function (defining location)
  of     Bytes
Calls  Allocated

 1542   6309568  __GI___libc_malloc (libc-2.17.so)
    1        72  __realloc (libc-2.17.so)


openss>>expview -vleaked,fullstack mem5

Number  Call Stack Function (defining location)
  of
Leaks
                _start (matmul)
                > @ 562 in __libc_start_main (libmonitor.so.0.0.0: main.c,541)
                >>__libc_start_main (libc-2.17.so)
                >>> @ 164 in main (matmul: matmul.c,161)
                >>>> @ 144 in do_work (matmul: matmul.c,137)
                >>>>> @ 51 in allocateMatrix (matmul: matmul.c,45)
  512           >>>>>>__GI___libc_malloc (libc-2.17.so)
                _start (matmul)
                > @ 562 in __libc_start_main (libmonitor.so.0.0.0: main.c,541)
                >>__libc_start_main (libc-2.17.so)
                >>> @ 164 in main (matmul: matmul.c,161)
                >>>> @ 145 in do_work (matmul: matmul.c,137)
                >>>>> @ 51 in allocateMatrix (matmul: matmul.c,45)
  512           >>>>>>__GI___libc_malloc (libc-2.17.so)
                _start (matmul)
                > @ 562 in __libc_start_main (libmonitor.so.0.0.0: main.c,541)
                >>__libc_start_main (libc-2.17.so)
                >>> @ 164 in main (matmul: matmul.c,161)
                >>>> @ 149 in do_work (matmul: matmul.c,137)
                >>>>> @ 51 in allocateMatrix (matmul: matmul.c,45)
  512           >>>>>>__GI___libc_malloc (libc-2.17.so)
                _start (matmul)
                > @ 562 in __libc_start_main (libmonitor.so.0.0.0: main.c,541)
                >>__libc_start_main (libc-2.17.so)
                >>> @ 164 in main (matmul: matmul.c,161)
                >>>> @ 144 in do_work (matmul: matmul.c,137)
                >>>>> @ 50 in allocateMatrix (matmul: matmul.c,45)
    1           >>>>>>__GI___libc_malloc (libc-2.17.so)
                _start (matmul)
                > @ 562 in __libc_start_main (libmonitor.so.0.0.0: main.c,541)
                >>__libc_start_main (libc-2.17.so)
                >>> @ 164 in main (matmul: matmul.c,161)
```

>>>> @ 145 in do_work (matmul: matmul.c,137)
>>>>> @ 50 in allocateMatrix (matmul: matmul.c,45)
1 >>>>>>__GI___libc_malloc (libc-2.17.so)

OpenSpeedShop Tutorial Hands-on EXERCISE-8

Using A OpenSpeedShop Memory (mem) Experiment to aid in Understanding Importance of the Usage of memory in application development.

Exercise 8 Assignment:

In this exercise we examine two SHOC CUDA benchmark tests to illustrate the capabilities of the OpenSpeedShop CUDA data gathering (osscuda), command line interface (CLI) views, and the new cuda focused graphical user interface (openss-gui).

1) Run the GEMM benchmark with OpenSpeedShop by running the PBS job script:

```
qsub run_cuda_GEMM_qsub.sh
```

2) You can use:

```
qstat -u <login id>
```

to monitor the job progress.

3) Once the job completes (qstat -u <login id> runs a blank line), then use the cuda centric graphical user interface to view the database file.

```
openss-gui -f GEMM-cuda-0.openss
```

4) Viewing the cuda performance data with the GUI display:

- Some basic information:
  - The default view for the CUDA centric GUI contains four main panes.
  - A time line tracking the cuda kernel executions are shown with a green color and data transfers to the GPU device and the CPU are shown with a red color. This is too give insight into the relative cost of the transfers versus the actual time spent executing in the kernel.
  - The shaded areas in the timeline represent performance information from the execution of the application in the GPU and in the CPU. So, one can infer if the GPU and CPU are being fully utilized.
  - The left pane, labeled "Loaded Experiments" is the area where one will be able to eventually select what process, thread, or rank performance information will be displayed in the timeline and metric value display panel.
    - Currently, it shows information about the application and hosts (processes, threads, ranks) that are involved in this GUI display.
  - The metric panel (lower right display panel) is where the text based performance information is displayed. There are options based on metric type that control the data being presented in this panel.

- A source view panel is also available by opening a GUI pane below the metric panel.

- Examine the GEMM database file and the performance information it contains.
    - Swap the Metric (exec_time) to (xfer_time) to see the data transfer summary of calls.
    - Swap the Mode from (Metric) to (Details) to see the chorological list of all kernel executions and data transfers.

- Examine the time line which represents the chronological list of kernel executions and data transfers in a graphical view.
    - Are there areas where the data transfers take more time than the kernel executions, i.e. more red colored lines than green?
    - This would mean that it might not be profitable to use the GPU for that section of the code.

- Slide the timeline around to zoom in an out and slide to the left or right in order to see different parts of the time line.

- Repeat this for the FFT database, FFT-cuda-0.openss

5) Viewing the cuda performance data with the CLI display:

- Because the CLI commands need more explanation, a hints file for each application was created with an explanation of the commands and the actual output from a previous run to illustrate what the outputs should be.

- See the Exercise 8 Hints for CLI Analysis of GEMM OpenSpeedShop database section for using the CLI to view the GEMM cuda performance data.

Exercise 8 Hints for CLI Analysis of GEMM OpenSpeedShop database

Here we use the CLI to examine the cuda experiment output for the GEMM cuda application

- expview shows a summary of the cuda kernel calls
    - openss>>expview

- expview -vxfer shows a summary of the cuda transfer (device to memory, etc.) calls
    - openss>>expview -vxfer

- expview -vtrace,exec shows a chronological list of the cuda kernel calls
    - openss>>expview -vtrace,exec

- `expview -vtrace,xfer shows a chronological list of the cuda transfer calls`
  - `openss>>expview -vxfer,trace`

- `expview -vfullstack,exec shows the call paths involving the cuda kernel calls`
  - `openss>>expview -vfullstack`

- `expview -vfullstack,xfer shows the call paths involving the cuda transfer calls`
  - `openss>>expview -vfullstack,xfer`

- `expview -vhwpc displays the sampled CPU/GPU hardware performance counters as a function of time. I.e. it does not display data as a function of source code constructs. Thus only the '-v Summary' and '-v SummaryOnly' options apply.`
  - `openss>>expview -vhwpc`

```
$ openss -cli -f GEMM-cuda-0.openss
openss>>[openss]: The restored experiment identifier is:  -x 1
openss>>expview

Exclusive      % of  Exclusive  Function (defining location)
Time (ms)     Total     Count
        Exclusive
           Time
 9.860058  52.092957       200  void RunTest<float>(std::string, ResultDatabase&, OptionParser&) (GEMM:
GEMM.cpp,156)
 9.067756  47.907043       200  void RunTest<double>(std::string, ResultDatabase&, OptionParser&) (GEMM:
GEMM.cpp,156)
openss>>expview -vxfer

Exclusive      % of  Exclusive  Function (defining location)
Time (ms)     Total     Count
        Exclusive
           Time
 1.184482  75.198904        46  void RunTest<float>(std::string, ResultDatabase&, OptionParser&) (GEMM:
GEMM.cpp,156)
 0.390650  24.801096        46  void RunTest<double>(std::string, ResultDatabase&, OptionParser&) (GEMM:
GEMM.cpp,156)

openss>>expview -vtrace,exec cuda20

  Start Time (d:h:m:s)  Exclusive      % of   Grid    Block  Call Stack Function (defining location)
                 Time (ms)     Total   Dims     Dims
                       Exclusive
                          Time
2016/11/09 13:24:33.642   0.055619   0.293848  4,4,1  16,16,1  >>void RunTest<float>(std::string,
ResultDatabase&, OptionParser&) (GEMM: GEMM.cpp,156)
2016/11/09 13:24:33.642   0.051906   0.274231  4,4,1  16,16,1  >>void RunTest<float>(std::string,
ResultDatabase&, OptionParser&) (GEMM: GEMM.cpp,156)
```

2016/11/09 13:24:33.642 0.052290 0.276260 4,4,1 16,16,1 >>void RunTest<float>(std::string, ResultDatabase&, OptionParser&) (GEMM: GEMM.cpp,156)
2016/11/09 13:24:33.643 0.051778 0.273555 4,4,1 16,16,1 >>void RunTest<float>(std::string, ResultDatabase&, OptionParser&) (GEMM: GEMM.cpp,156)
2016/11/09 13:24:33.643 0.052130 0.275415 4,4,1 16,16,1 >>void RunTest<float>(std::string, ResultDatabase&, OptionParser&) (GEMM: GEMM.cpp,156)
2016/11/09 13:24:33.643 0.051810 0.273724 4,4,1 16,16,1 >>void RunTest<float>(std::string, ResultDatabase&, OptionParser&) (GEMM: GEMM.cpp,156)
2016/11/09 13:24:33.644 0.051523 0.272208 4,4,1 16,16,1 >>void RunTest<float>(std::string, ResultDatabase&, OptionParser&) (GEMM: GEMM.cpp,156)
2016/11/09 13:24:33.644 0.051843 0.273899 4,4,1 16,16,1 >>void RunTest<float>(std::string, ResultDatabase&, OptionParser&) (GEMM: GEMM.cpp,156)
2016/11/09 13:24:33.645 0.051778 0.273555 4,4,1 16,16,1 >>void RunTest<float>(std::string, ResultDatabase&, OptionParser&) (GEMM: GEMM.cpp,156)
2016/11/09 13:24:33.645 0.051586 0.272541 4,4,1 16,16,1 >>void RunTest<float>(std::string, ResultDatabase&, OptionParser&) (GEMM: GEMM.cpp,156)
2016/11/09 13:24:33.646 0.052163 0.275589 4,4,1 16,16,1 >>void RunTest<float>(std::string, ResultDatabase&, OptionParser&) (GEMM: GEMM.cpp,156)
2016/11/09 13:24:33.647 0.052066 0.275077 4,4,1 16,16,1 >>void RunTest<float>(std::string, ResultDatabase&, OptionParser&) (GEMM: GEMM.cpp,156)
2016/11/09 13:24:33.648 0.051554 0.272372 4,4,1 16,16,1 >>void RunTest<float>(std::string, ResultDatabase&, OptionParser&) (GEMM: GEMM.cpp,156)
2016/11/09 13:24:33.648 0.051811 0.273729 4,4,1 16,16,1 >>void RunTest<float>(std::string, ResultDatabase&, OptionParser&) (GEMM: GEMM.cpp,156)
2016/11/09 13:24:33.649 0.052099 0.275251 4,4,1 16,16,1 >>void RunTest<float>(std::string, ResultDatabase&, OptionParser&) (GEMM: GEMM.cpp,156)
2016/11/09 13:24:33.649 0.052194 0.275753 4,4,1 16,16,1 >>void RunTest<float>(std::string, ResultDatabase&, OptionParser&) (GEMM: GEMM.cpp,156)
2016/11/09 13:24:33.650 0.051587 0.272546 4,4,1 16,16,1 >>void RunTest<float>(std::string, ResultDatabase&, OptionParser&) (GEMM: GEMM.cpp,156)
2016/11/09 13:24:33.653 0.052483 0.277280 4,4,1 16,16,1 >>void RunTest<float>(std::string, ResultDatabase&, OptionParser&) (GEMM: GEMM.cpp,156)
2016/11/09 13:24:33.656 0.052195 0.275758 2,1,1 8,32,1 >>void RunTest<double>(std::string, ResultDatabase&, OptionParser&) (GEMM: GEMM.cpp,156)
2016/11/09 13:24:33.659 0.051842 0.273893 4,4,1 16,16,1 >>void RunTest<float>(std::string, ResultDatabase&, OptionParser&) (GEMM: GEMM.cpp,156)

openss>>expview -vxfer,trace cuda15

Start Time (d:h:m:s)	Exclusive Time (ms)	% of Total Exclusive Time	Size	Kind	Call Stack Function (defining location)
2016/11/09 13:24:33.641	0.027809	1.765503	262144	HostToDevice	>>void RunTest<float>(std::string, ResultDatabase&, OptionParser&) (GEMM: GEMM.cpp,156)
2016/11/09 13:24:33.641	0.027362	1.737124	262144	HostToDevice	>>void RunTest<float>(std::string, ResultDatabase&, OptionParser&) (GEMM: GEMM.cpp,156)
2016/11/09 13:24:33.643	0.026818	1.702587	262144	DeviceToHost	>>void RunTest<float>(std::string, ResultDatabase&, OptionParser&) (GEMM: GEMM.cpp,156)
2016/11/09 13:24:33.644	0.026786	1.700556	262144	DeviceToHost	>>void RunTest<float>(std::string, ResultDatabase&, OptionParser&) (GEMM: GEMM.cpp,156)

2016/11/09 13:24:33.645 0.026818 1.702587 262144 DeviceToHost >>void RunTest<float>(std::string, ResultDatabase&, OptionParser&) (GEMM: GEMM.cpp,156)
2016/11/09 13:24:33.646 0.026786 1.700556 262144 DeviceToHost >>void RunTest<float>(std::string, ResultDatabase&, OptionParser&) (GEMM: GEMM.cpp,156)
2016/11/09 13:24:33.651 0.027042 1.716808 262144 DeviceToHost >>void RunTest<float>(std::string, ResultDatabase&, OptionParser&) (GEMM: GEMM.cpp,156)
2016/11/09 13:24:33.652 0.026818 1.702587 262144 DeviceToHost >>void RunTest<float>(std::string, ResultDatabase&, OptionParser&) (GEMM: GEMM.cpp,156)
2016/11/09 13:24:33.653 0.027809 1.765503 262144 HostToDevice >>void RunTest<float>(std::string, ResultDatabase&, OptionParser&) (GEMM: GEMM.cpp,156)
2016/11/09 13:24:33.653 0.027362 1.737124 262144 HostToDevice >>void RunTest<float>(std::string, ResultDatabase&, OptionParser&) (GEMM: GEMM.cpp,156)
2016/11/09 13:24:33.654 0.026818 1.702587 262144 DeviceToHost >>void RunTest<float>(std::string, ResultDatabase&, OptionParser&) (GEMM: GEMM.cpp,156)
2016/11/09 13:24:33.656 0.026849 1.704556 262144 DeviceToHost >>void RunTest<float>(std::string, ResultDatabase&, OptionParser&) (GEMM: GEMM.cpp,156)
2016/11/09 13:24:33.657 0.026786 1.700556 262144 DeviceToHost >>void RunTest<float>(std::string, ResultDatabase&, OptionParser&) (GEMM: GEMM.cpp,156)
2016/11/09 13:24:33.658 0.027073 1.718777 262144 DeviceToHost >>void RunTest<float>(std::string, ResultDatabase&, OptionParser&) (GEMM: GEMM.cpp,156)
2016/11/09 13:24:33.661 0.026817 1.702524 262144 DeviceToHost >>void RunTest<float>(std::string, ResultDatabase&, OptionParser&) (GEMM: GEMM.cpp,156)

openss>>expview -vfullstack

Exclusive Time (ms) % of Total Exclusive Time Count Call Stack Function (defining location)
main (GEMM: main.cpp,136)
> @ 293 in RunBenchmark(ResultDatabase&, OptionParser&) (GEMM: GEMM.cpp,123)
7.864320 41.549014 160 >> @ 240 in void RunTest<float>(std::string, ResultDatabase&, OptionParser&) (GEMM: GEMM.cpp,156)
main (GEMM: main.cpp,136)
> @ 293 in RunBenchmark(ResultDatabase&, OptionParser&) (GEMM: GEMM.cpp,123)
7.237557 38.237680 160 >> @ 240 in void RunTest<double>(std::string, ResultDatabase&, OptionParser&) (GEMM: GEMM.cpp,156)
main (GEMM: main.cpp,136)
> @ 293 in RunBenchmark(ResultDatabase&, OptionParser&) (GEMM: GEMM.cpp,123)
1.995738 10.543943 40 >> @ 231 in void RunTest<float>(std::string, ResultDatabase&, OptionParser&) (GEMM: GEMM.cpp,156)
main (GEMM: main.cpp,136)
> @ 293 in RunBenchmark(ResultDatabase&, OptionParser&) (GEMM: GEMM.cpp,123)
1.830199 9.669363 40 >> @ 231 in void RunTest<double>(std::string, ResultDatabase&, OptionParser&) (GEMM: GEMM.cpp,156)

openss>>expview -vfullstack,xfer

Exclusive Time (ms) % of Total Exclusive Time Count Call Stack Function (defining location)

main (GEMM: main.cpp,136)
> @ 293 in RunBenchmark(ResultDatabase&, OptionParser&) (GEMM: GEMM.cpp,123)
1.071674 68.037091 40 >> @ 250 in void RunTest<float>(std::string, ResultDatabase&, OptionParser&)
(GEMM: GEMM.cpp,156)
main (GEMM: main.cpp,136)
> @ 293 in RunBenchmark(ResultDatabase&, OptionParser&) (GEMM: GEMM.cpp,123)
0.329138 20.895900 40 >> @ 250 in void RunTest<double>(std::string, ResultDatabase&, OptionParser&)
(GEMM: GEMM.cpp,156)
main (GEMM: main.cpp,136)
> @ 293 in RunBenchmark(ResultDatabase&, OptionParser&) (GEMM: GEMM.cpp,123)
0.055618 3.531006 2 >> @ 195 in void RunTest<float>(std::string, ResultDatabase&, OptionParser&)
(GEMM: GEMM.cpp,156)
main (GEMM: main.cpp,136)
> @ 293 in RunBenchmark(ResultDatabase&, OptionParser&) (GEMM: GEMM.cpp,123)
0.054724 3.474249 2 >> @ 197 in void RunTest<float>(std::string, ResultDatabase&, OptionParser&)
(GEMM: GEMM.cpp,156)
main (GEMM: main.cpp,136)
> @ 293 in RunBenchmark(ResultDatabase&, OptionParser&) (GEMM: GEMM.cpp,123)
0.030051 1.907840 2 >> @ 195 in void RunTest<double>(std::string, ResultDatabase&, OptionParser&)
(GEMM: GEMM.cpp,156)
main (GEMM: main.cpp,136)
> @ 293 in RunBenchmark(ResultDatabase&, OptionParser&) (GEMM: GEMM.cpp,123)
0.029187 1.852988 2 >> @ 197 in void RunTest<double>(std::string, ResultDatabase&, OptionParser&)
(GEMM: GEMM.cpp,156)
main (GEMM: main.cpp,136)
> @ 293 in RunBenchmark(ResultDatabase&, OptionParser&) (GEMM: GEMM.cpp,123)
0.002466 0.156558 2 >> @ 503 in void RunTest<float>(std::string, ResultDatabase&, OptionParser&)
(GEMM: GEMM.cpp,156)
main (GEMM: main.cpp,136)
> @ 293 in RunBenchmark(ResultDatabase&, OptionParser&) (GEMM: GEMM.cpp,123)
0.002274 0.144369 2 >> @ 503 in void RunTest<double>(std::string, ResultDatabase&, OptionParser&)
(GEMM: GEMM.cpp,156)

```
openss>>expview -vhwpc

Time    CPU All    GPU All   |<-------------CPU---|---GPU------------>|
(ms)
   0   12053790          0   |                   **|                    |
  11    5662505          0   |                    *|                    |
  22    4905899          0   |                    *|                    |
  33    4776648          0   |                    *|                    |
  44    4309918          0   |                    *|                    |
  55    5640856          0   |                    *|                    |
  66    6546501          0   |                    *|                    |
  77    2234152          0   |                     |                    |
  88    2234152          0   |                     |                    |
  99    1900781          0   |                     |                    |
 110     859674          0   |                     |                    |
 121     123935          0   |                     |                    |
 132     111020          0   |                     |                    |
 143     128966          0   |                     |                    |
 154     139401          0   |                     |                    |
 165     141358          0   |                     |                    |
```

```
 176    148326         0  |                    |                     |
 187    174903         0  |                    |                     |
 198    199007         0  |                    |                     |
 209    191441         0  |                    |                     |
 220    186662         0  |                    |                     |
 231    196182         0  |                    |                     |
 242    206272         0  |                    |                     |
 253    198724         0  |                    |                     |
 264   1500617         0  |                    |                     |
 275   8187823         0  |                   *|                     |
 286  45158790         0  |          **********|                     |
 297  83613456         0  |********************|                     |
 308  82094389         0  | ********************|                     |
 319  64889504         0  |     ***************|                     |
 330  75615859         0  |  ******************|                     |
 341  72513400         0  |   *****************|                     |
 352  72083823         0  |   *****************|                     |
 363  74116133         0  |   *****************|                     |
 374  80815756         0  | ********************|                     |
 385  75330132         0  |  ******************|                     |
 396  67893390         0  |    ****************|                     |
 407  69458982         0  |    ****************|                     |
 418  66882493  13128909  |     ***************|*********************|
 429  79384665  12124825  |  ******************|******************   |
 440  79603739   8823487  | ********************|*************        |
 451  17761878   2277979  |                 ****|***                  |
 462         0         0  |                    |                     |
 473         0         0  |                    |                     |
openss>>quit
```