For the first assignment, you will run a benchmark called `Stream2` on the Stampede2 system and on another system of your choosing. This benchmark runs several tests to measure the bandwidth of the different memory components (L1, L2, L3 cache, main DRAM, etc.). This benchmark is based on work originally done by John D. McCalpin (aka, *Dr. Bandwidth*) but I have modified it with a more accurate timer and using C/C++ instead of the original Fortran.

Part (1)

Pull the COMP 364/464 F16 source from BitBucket. The easiest approach is to clone using Git from Stampede2:

```
prompt% git clone https://cstone2@bitbucket.org/cstone2/hpc_f17.git
```

You can also clone from a repository fork which is the best approach since you can push/pull from that and use that to submit your source code. By now, all students should have received and accepted an invitation to the repository.

The stream benchmark is in `hpc_f16/hw1`. The `Makefile` is configured for Stampede2 but should work on just about any Linux (or Unix-like) system. It's best to compile on an actual compute node so that the compiler can detect what type of hardware is available. (Otherwise, you'd have to specific other options on the command-line.) So, let's request an interactive node on Stampede2:

On Stampede2, request a dedicated node by executing idev:

```
prompt% idev -p flat-quadrant
```

This will automatically generate a request to the queue system to provide you with an interactive session on a dedicated node with the Flat-Quadrant memory configuration. (It may ask you about the # of tasks; If so, enter 68.) Once successful, the queue system will log you into a dedicated node. The session will last for a maximum of 30 minutes so don't dally. Note that it can take anywhere from 30 seconds to 20 minutes (or more) for the queue system to respond with a dedicated node. If you are unable to get an interactive node, you can configure a batch session. See the Stampede2 user guide for instructions and/or ask me for assistance.

Once you have an interactive session, compile `stream2.cpp by` launching make from inside `hpc_f16/hw1`:

```
prompt% make
```

Again, this "should" work on any Linux-like machine with a functioning C++ compiler. The default compiler is the GNU C++ compiler (g++). We'll switch compilers shortly.

Next, run the *stream2* executable with the default settings. You should get a report (comma separated variables) with something like the following:

```
prompt% ./stream2

Smallest detectable time = 1.000000e-06 (ms)
getTicksPerSecond = 1.448467e+09
Size (iterations), Fill (MB/s), Copy, AXPY, DOT, Uncertainty (%)
30, 14982.497656, 18169.690970, 30982.351087, 13101.115888, 6.5%
```

```
41, 19135.125009, 23118.084843, 33103.316384, 17093.253683, 6.1%
56, 24541.959725, 31177.092390, 43803.886910, 23982.201376, 5.7%
77, 31533.329489, 34627.489927, 57445.403922, 24469.417271, 5.4%
106, 39601.066199, 43852.070961, 70796.091616, 29273.844487, 4.9%
145, 42884.835134, 43822.977847, 75734.655676, 36179.375388, 3.9%
199, 56433.263005, 51918.643749, 83851.754370, 43059.409824, 3.7%
273, 58031.628805, 59878.759720, 90494.143683, 38379.753772, 2.8%
375, 69879.461575, 68797.244305, 119487.665833, 43319.220412, 2.4%
514, 79527.450153, 76479.803176, 122530.166578, 41380.559145, 2.0%
704, 82116.523331, 81197.341040, 134964.076455, 45633.141897, 1.5%
965, 95537.843808, 81279.477954, 150192.123888, 51074.162886, 1.3%
1324, 106442.097101, 84034.053082, 166393.333906, 54021.250104, 1.1%
1815, 111830.749623, 69917.632343, 173067.055927, 56657.030147, 0.8%
2488, 114998.396605, 38063.968781, 90910.816288, 56475.059332, 0.6%
3412, 116845.677363, 35612.232346, 83188.364337, 57224.939539, 0.4%
4678, 35282.884718, 34900.880825, 83186.106239, 57794.637215, 0.1%
6413, 35596.413521, 35393.066863, 82949.353882, 57735.717804, 0.1%
8793, 35600.817110, 35784.159117, 83907.614064, 58689.372267, 0.1%
12056, 36069.359289, 36093.170309, 85495.772552, 60418.261617, 0.0%
16530, 35719.828666, 36000.878542, 84097.057899, 59885.873992, 0.0%
22663, 35758.894901, 36093.919507, 85706.918350, 60516.929382, 0.0%
31072, 35713.372730, 36147.849736, 84877.976926, 60274.379404, 0.0%
42602, 34097.184289, 35665.555652, 81667.547697, 59690.123959, 0.0%
58410, 32959.335208, 26386.071557, 60036.634029, 33920.970176, 0.0%
80084, 31669.902144, 19662.857150, 39387.583781, 16159.733984, 0.0%
109800, 22880.202612, 19349.942612, 31923.846049, 13697.656530, 0.0%
150542, 18329.264051, 17121.856351, 34005.112113, 12984.779112, 0.0%
206403, 16874.993410, 16651.396773, 33164.334176, 12536.820893, 0.0%
282991, 12418.887890, 17121.952276, 25430.729989, 12660.127681, 0.0%
387998, 16836.005466, 17232.293945, 33990.215113, 12771.676565, 0.0%
531969, 8521.064043, 17086.178153, 17420.347468, 12832.615243, 0.0%
729362, 8480.660726, 17051.775434, 16904.450857, 12544.131325, 0.0%
1000000, 8657.004318, 17254.700682, 16783.227766, 12660.441358, 0.0%
```

The columns are the # of loop iterations, Fill, Copy, DAXPY, DDOT, and Uncertainty metrics. (We'll discuss these in class.) The values for Fill, Copy, Axpy, Dot are all bandwidth in MB/s.

Once that is successful, gather the CPU properties (frequency, model), memory (cache and main memory sizes), OS, and compiler. This can be found with `lscpu`, `free`, `uname`, and inspecting `/proc/cpuinfo` on Linux.

Now, rerun the program with a large enough maximum array size to make sure that the last-level cache is stressed. A good guide is to run with a maximum array size 10-20 times larger than the last-level cache size. This can be changed on the command line:

```
prompt% ./stream2.exe -max <maximum array size>
```

The default values are min = 30, max = 1,000,000, # samples = 32, # iterations = 50. We want to make sure to stress all of the cache levels and main memory. Determine if the default values are large enough already. Note that this program uses doubles for all array values. You can approximate the array sizes

based on this knowledge. Also, run with a larger # of samples and iterations to make a more complete dataset.

Part (2)

Repeat (1) using the Intel C++ compiler. Recall, the default is the GNU C++ compiler. You can change this on the command line as such:

```
prompt% make clean
prompt% CXX=icpc make
```

Part (3)

Plot the bandwidth for the four tests as a function of the loop length with both compilers. Also plot the uncertainty (use a different axis). Write a brief summary of your results discussing the bandwidth as a function of size. Review the source code for the four tests and determine the number of memory transactions and floating-point operations that are needed each iteration. Use this information to discuss the difference bandwidths measured on Stampede2 for each test. How do these change on your selected system and what caused this difference?

Submission:

Include in your report the screen output from the tests, the plot of the bandwidths, the system details, and your discussion. Submit to Sakai.

You are encouraged, but not required, to work in teams of 2. Only one report submission is needed per team. Include both student's names on the report. If you are not the student submitting the report, you must still submit a note on Sakai stating with whom you partnered.

Extra Credit (10%):

The Intel Xeon Phi Knights Landing (KNL) processor has a few configuration options. I had you select the configuration known as flat-quadrant. What did this mean? The KNL has two types of "main" memory, normal DDR4 RAM (96 GB) and MCDRAM (16 GB). In flat-quadrant, the MCDRAM is viewed a separate. To use this, you need to either allocation from this region of memory specially (not easy) or tell the OS to allocate everything from there. If you hadn't selected flat-quadrant, you would have been placed on a default node with cache-quadrant which means the MCDRAM acts like a huge L3 cache (with about 15 GB). Rerun the benchmarks above on either a cache-quadrant device or learn how to use the tool `numactl` to allocate on the MCDRAM. (The Stampede2 documentation section 'Managing memory' has that information.)