## LAB 1: GPU Matrix Transposition and Multiplication

* A PDF report documenting your work which discusses your findings carefully and offers insights. The report should present a clear evaluation of the design of your code and an analysis of performance including bottlenecks. Discuss what you did to improve the performance, and what worked and what did not work, and why. If the causes are not clear, offer your best guesses. Use plots and tables to report performance in terms of GFLOP/s or memory bandwidth.
* Your code and instructions for building. We should be able to just type make to build. Provide clear instructions to run the executables.

**Part 1: Matrix transpose optimization (TO be done on LONESTAR5)**

Optimize the CUDA matrix transpose implementations in transpose\_cuda.cu. Read ALL of the TODO comments in transpose\_device.cu. Matrix transpose is a common exercise in GPU optimization, so do not search for existing GPU matrix transpose code on the internet.

Your transpose code only need to be able to transpose square matrices where the side length is a multiple of 64.

The initial implementation has each block of 1024 threads handle a 64x64 block of the matrix, but you can change anything about the kernel if it helps obtain better performance.

The main method of transpose.cc already checks for correctness for all transpose results, so there should be an assertion failure if your kernel produces incorrect output.

The purpose of the shmemTransposeKernel is to demonstrate proper usage of global and shared memory. The optimalTransposeKernel should be built on top of shmemTransposeKernel and should incorporate any "tricks" such as ILP, loop unrolling, and anything that you think may be helpful to achieve higher performance.

You can compile and run the code by running

make transpose   
./transpose