ME 759
High Performance for Engineering Applications
Assignment 5
Due Thursday 2/27/2020 at 9:00 PM

Submit responses to all tasks which don't specify a file name to Canvas in a file called assignment5.txt, docx, pdf, rtf, odt (choose one of the formats). Also all plots should be submitted on Canvas. All *source files* should be submitted in the `HW05` subdirectory on the `master` branch of your homework git repo with no subdirectories.

All commands or code must work on *Euler* with only the `cuda` module loaded unless specified otherwise. They may behave differently on your computer, so be sure to test on Euler before you submit.

Please submit clean code. Consider using a formatter like clang-format.
* Before you begin, copy the provided files from `HW05` of the ME759-2020 repo.

1. a) Implement in a file called `reduce.cu` the functions `reduce` and `reduce_kernel` as declared and described in `reduce.cuh`. Your `reduce_kernel` should use the alteration from reduction #3 ("sequential addressing" from Lecture 13) and `reduce` should wrap calls to `reduce_kernel`.
   b) Write a test program `task1.cu` which does the following.
      - Creates and fills however you like an array of length `N` where `N` is the first command line argument as below.
      - Uses your `reduce` to sum the elements in the array. Uses the `threads_per_block` from the second command line argument as below.
      - Prints the resulting sum.
      - Prints the time taken to run the reduction in *milliseconds*.
      - Compile: `nvcc task1.cu reduce.cu -Xcompiler -O3 -Xcompiler -Wall -Xptxas -O3 -o task1`
      - Run (where `N` $\leq 2^{30}$ and `threads_per_block` are positive integers):
        `./task1 N threads_per_block`
      - Exampled expected output:
        `102536`
        `1031.2`
   c) On an Euler *compute node*, run `task1` for each value `n` $= 2^{10}, 2^{11}, \cdots, 2^{30}$ and generate plot the time taken by your algorithm as a function of `N` when `threads_per_block` $=$ 1024. Overlay another plot which plots the same relationship with a different choice of `threads_per_block`.

2. a) Implement in a file called `matmul.cu` the functions `matmul` and `matmul_kernel` as declared and described in `matmul.cuh`. Be sure to follow the use of shared memory tiles. These should be based on the tiled matrix multiplication method presented in Lecture 11. Your implementation should work for arbitrary matrix dimension `n` $\leq 2^{15}$.

   b) Write a test program `task2.cu` which does the following.

   - Creates and fills however you like row-major representations of `n`×`n` matrices `A`, `B`, and `C` in managed memory, where `n` is the first command line argument as below.
   - Uses your `matmul` function to produce `C` as the matrix product `AB`.
   - Prints the first element of the resulting `C`.
   - Prints the last element of the resulting `C`.
   - Prints the time taken to run the matrix multiplication in *milliseconds* using CUDA events.
   - Compile: `nvcc task2.cu matmul.cu -Xcompiler -O3 -Xcompiler -Wall -Xptxas -O3 -o task2`
   - Run (where `n` and `block_dim` are positive integers and `n` is not necessarily a multiple of `block_dim`): `./task2 n block_dim`
   - Exampled expected output:
     ```
     1025.1
     561.3
     10256.2
     ```

   c) On an Euler *compute node*, run `task2` for each value `n` $= 2^5, 2^6, \cdots, 2^{15}$ and generate a plot of the time taken by your algorithm as a function of `n`.

   d) What is the best performing value of `block_dim` when `n` $= 2^{15}$?

   e) Present the best runtime for `n` $= 2^{15}$ from your HW04 matrix multiplication task (naive GPU implementation). Explain why the tiled approach performs better.

   f) Present the runtime for `n` $= 2^{15}$ from HW02 (serial implementation `mmul1`) (or state that it goes beyond the Euler time limit). Explain why both GPU implementations perform better.