ME 759
High Performance for Engineering Applications
Assignment 3
Due Thursday 02/13/2020 at 9:00 PM

Submit responses to all tasks which don't specify a file name to Canvas in a file called assignment3.txt, docx, pdf, rtf, odt (choose one of the formats). All *source files* should be submitted in the `HW03` subdirectory on the `master` branch of your CAE git repo with no subdirectories.

All commands or code must work on *Euler* with only the `cuda` module loaded unless specified otherwise. Commands and/or code 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 `HW03` of the ME759-2020 repo. Do not change any of the provided files because we will write clean copies over them when grading.

1. Write a C++ program using CUDA in a file called `task1.cu` which launches a GPU kernel with 1 block and 4 threads. Inside the kernel, each thread should use `std::printf` to write out `Hello World!  I am thread x.` (followed by a newline), where `x` is the index of the thread that prints the message. (Follow your kernel call with a call to `cudaDeviceSynchronize()` so that the host waits for the kernel to finish printing before returning from `main`.)

   - Compile: `nvcc task1.cu -Xcompiler -O3 -Xcompiler -Wall -Xptxas -O3 -o task1`
   - Run: `./task1`
   - Expected output (lines could be out of order):
     ```
     Hello World!  I am thread 0.
     Hello World!  I am thread 1.
     Hello World!  I am thread 2.
     Hello World!  I am thread 3.
     ```

2. Write a C++ program using CUDA in a file called `task2.cu` which does the following:

   - From the host, allocates an array of 16 `int`s on the device called `dA`.
   - Launches a kernel with 2 blocks, each block having 8 threads. Each thread computes the sum of its thread index and its block index and writes the result into `dA`.
   - Copies back the data stored in the device array `dA` into a host array called `hA`.
   - Prints (from the host) the 16 values stored in the host array separated by a single space each.

   - Compile: `nvcc task2.cu -Xcompiler -O3 -Xcompiler -Wall -Xptxas -O3 -o task2`
   - Run: `./task2`
   - Expected output (followed by newline): `0 1 2 3 4 5 6 7 1 2 3 4 5 6 7 8`

3. a) Implement in a file called `vadd.cu`, the `vadd` kernel function as declared and described in `vadd.cuh`. This function should take in two arrays, `a` and `b`, and add them together, storing the result in `b`. Each thread should do at most one of the additions, adding the right element from `a` with the corresponding element from `b`.

   b) Write a file `task3.cu` which does the following.

   - Creates two arrays of length `n` where `n` is read from the first command line argument with whatever values you like.
   - Calls your `vadd` kernel with a 1D execution configuration that uses 512 threads per block.
   - Prints the amount of time taken to execute the kernel in *seconds* using CUDA events[1].
   - Prints the first element of the resulting array.
   - Prints the last element of the resulting array.
   - Compile: `nvcc task3.cu vadd.cu -Xcompiler -O3 -Xcompiler -Wall -Xptxas -O3 -o task3`
   - Run (where `n` is a positive integer): `./task3 n`
   - Example expected output (followed by newline):
     ```
     0.2
     35.3
     1.3
     ```

   c) On an Euler *compute node*, run `task3` for each value $\texttt{n} = 2^{10}, 2^{11}, \cdots, 2^{30}$ and generate a plot `task3.pdf` which plots the time taken by your `vadd` as a function of `n`. Overlay another plot which shows the scaling results when using 1024 threads per block.

[1] Recall the document `timing.md`.