ECE 759
High Performance Computing for Engineering Applications
Assignment 5
Due Friday 11/08/2024 at 23:59 PM

Submit responses to all tasks which don't specify a file name to Canvas in a file called `assignment5.pdf`. Submit all plots (if any) on Canvas. Do not zip your Canvas submission.

All *source files* should be submitted in the `HW05` subdirectory on the `main` branch of your GitLab repo. Please use the name `HW05` exactly as shown here (both in terms of capitalization & name). The `HW05` subdirectory should have no subdirectories. For this assignment, your `HW05` folder should contain `task1.cu`, `task2.cu`, `task3.cu`, and `vscale.cu`.

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

- `#SBATCH --gpus-per-task=1` should be added, which requests one node with 1 GPU.

Please submit clean code. Consider using a formatter like clang-format.

IMPORTANT: Before you begin, copy any provided files from `Assignments/HW05` directory of the ECE 759 Resource Repo. Do not change any of the provided files since these files will be overwritten with clean, reference copies when grading.

The GitHub link to your code folder should be: `https://github.com/YourGitHubName/repo759/HW05`

---

1. Write a C++ program using CUDA in a file called `task1.cu` which computes the factorial of integers from 1 to 8, by launching a GPU kernel with 1 block and 8 threads. Inside the kernel, each thread should use `std::printf` to write out `a!=b` (followed by a newline), where `a` is one of the 8 integers, and `b` is the result of `a!`. (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 -std=c++17 -o task1`
   - Run (on Euler, use Slurm sbatch!): `./task1`
   - Expected output (showing only 4 out of the 8 lines expected; lines could be out of order):
     ```
     1!=1
     2!=2
     3!=6
     4!=24
     ```
   - It is ok to publish your sbatch script on Piazza. For this assignment, you will need to ask Slurm to execute your program on a node that has GPU cards

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 `ax+y` and writes the result in one distinct entry of the `dA` array. Here,
     - `x` is the thread's `threadIdx`;
     - `y` is the thread's `blockIdx`;
     - `a` is an integer argument that the kernel takes (so all threads use the same `a`). You need to generate `a` randomly and then call the kernel with it. It is up to you how you generate this random number, one possible approach is described here BestPractice.
   - 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.

   How to go about it, and what the expected output looks like:

   - Compile: `nvcc task2.cu -Xcompiler -O3 -Xcompiler -Wall -Xptxas -O3 -std=c++17 -o task2`
   - Run (on Euler, use Slurm sbatch!): `./task2`
   - Expected output (followed by newline; yours could be different depending on the random number generation): `0 10 20 30 40 50 60 70 1 11 21 31 41 51 61 71`

3. a) Implement in a file called `vscale.cu`, the `vscale` kernel function as declared and described in `vscale.cuh`. This function should take in two arrays, `a` and `b`, and do an element-wise multiplication of the two arrays: $b_i = a_i \cdot b_i$. In the process, `b` gets overwritten. Each thread should do at most one of the multiplication operations.

   Example:

   $$\texttt{a} = [-5.0, 2.0, 1.5], \quad \texttt{b} = [0.8, 0.3, 0.6], \quad \texttt{n} = 3$$

   The resulting `b` array is:

   $$\texttt{b} = [-4.0, 0.6, 0.9].$$

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

   - Creates two arrays of length `n` filled by random numbers[1] where `n` is read from the first command line argument. The range of values for array `a` is [-10.0, 10.0], whereas the range of values for array `b` is [0.0, 1.0].
   - Calls your `vscale` kernel with a 1D execution configuration that uses 512 threads per block.
   - Prints the amount of time taken to execute the kernel in *milliseconds* using CUDA events[2].
   - Prints the first element of the resulting array.
   - Prints the last element of the resulting array.

   How to go about it, and what the expected output looks like:

   - Compile: `nvcc task3.cu vscale.cu -Xcompiler -O3 -Xcompiler -Wall -Xptxas -O3 -std=c++17 -o task3`
   - Run (where `n` is a positive integer): `./task3 n`
   - Example expected output (followed by newline):
     ```
     0.012
     1.3
     2.3
     ```

   c) On an Euler *compute node*, run `task3` for each value $\texttt{n} = 2^{10}, 2^{11}, \cdots, 2^{29}$ and generate a plot `task3.pdf` which plots the time taken by your `vscale` as a function of `n`. Overlay another plot which shows the scaling results when using 16 threads per block.

[1] Details about random number generation can be found in random_numbers.md.
[2] Recall the GPU timing section of the document timing.md.