ECE 759
High Performance Computing for Engineering Applications
Assignment 6
Due Friday 4/14/2025 at 23:59 PM

Submit all plots on Canvas. Do not zip your Canvas submission. All *source files* should be stored in the `HW06` folder on the `main` branch of your `GitHub` repo. Please use the name `HW06` exactly as shown here (both in terms of capitalization & name). The `HW06` subdirectory should have no subdirectories.

For this assignment, your `HW06` folder should contain `task1.cu`, `task2.cu`, `matmul.cu`, and `stencil.cu`.

All commands or code must work on *Euler* with only the `nvidia/cuda` module loaded.
Since various commands may behave differently on your computer, we recommend that you test on *Euler* before you submit your homework.

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

IMPORTANT: Before you begin, copy any provided files from `Assignments/HW06` 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.

github.com/kagrawal6/repo759/tree/main/HW06

The GitHub link to your code folder should be: `https://github.com/YourGitHubName/repo759/HW06`

---

1. (a) Implement in a file called `matmul.cu` the `matmul` and `matmul_kernel` functions as declared and described in the comment section of `matmul.cuh`. These functions should compute the product of square matrices.

   (b) Write a program `task1.cu` which will complete the following (some memory management steps are omitted for clarity, but you should implement them in your code for it to work properly):
   - Create matrices (as 1D row major arrays) `A` and `B` of size `n` × `n` on the host.
   - Fill these matrices with random numbers in the range [-1, 1].
   - Prepare arrays that are allocated as device memory (they will be passed to your `matmul` function.)
   - Call your `matmul` function.
   - Print the last element of the resulting matrix.
   - Print the time taken to execute your `matmul` function in *milliseconds* using CUDA events.
   - Compile: `nvcc task1.cu matmul.cu -Xcompiler -O3 -Xcompiler -Wall -Xptxas -O3 -std c++17 -o task1`
   - The executable is to be run as: `./task1 n threads_per_block`
     - The values `n` and `threads_per_block` are positive integers
     - The matrix size `n` is not necessarily a power of 2. We will not test with sizes larger than 2^{14}. For the number of threads, we will use multiples of 32.
     - Use `Slurm` to run your job on Euler
   - Example expected output:
     `-16.35`
     `1.23`

   (c) On an Euler *compute node*, run `task1` for each value `n` $= 2^5, 2^6, \cdots, 2^{14}$ and generate a plot `task1.pdf` that reports 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 `threads_per_block` of your choice.

   (d) Going beyond the call of duty, do if you wish to: Compare the scaling results with the results obtained in a previous assignment where you did a similar scaling analysis using a sequential implementation on the CPU. If you do this, drop your plots on Piazza. Please comment on the noted results.

2. (a) Implement in a file called `stencil.cu` the `stencil` and `stencil_kernel` functions as declared and described in the comment section of `stencil.cuh`. These functions should produce the 1D convolution of `image` and `mask` as the following:

$$\texttt{output}[i] = \sum_{j=-R}^{R} \texttt{image}[i+j] * \texttt{mask}[j+R] \qquad i = 0, \cdots, \texttt{n} - 1 .$$

Assume that $\texttt{image}[i] = 1$ when $i < 0$ or $i > \texttt{n} - 1$. Please read the comments in the `stencil.cuh` file to see which variables should be stored in shared memory. The use of shared memory is an important component of this problem.

(b) Write a program `task2.cu` which will complete the following (some memory management steps are omitted for clarity, but you should implement them in your code):

- Create arrays `image` (length `n`), `output` (length `n`), and `mask` (length `2 * R + 1`) on the host.
- Fill the `image` and `mask` array with random numbers in the range [-1, 1].
- Prepare arrays that are allocated as device memory (they will be passed to your `stencil` function.)
- Call your `stencil` function.
- Print the last element of the resulting `output` array.
- Print the time taken to execute your `stencil` function in *milliseconds* using CUDA events.
- Compile: `nvcc task2.cu stencil.cu -Xcompiler -O3 -Xcompiler -Wall -Xptxas -O3 -std c++17 -o task2`
- Run via `Slurm` (where `n`, `R`, and `threads_per_block` are positive integers):
  `./task2 n R threads_per_block`
- Example expected output:
  `11.36`
  `1.23`

(c) On an Euler *compute node*, run `task2` for each value $\texttt{n} = 2^{10}, 2^{11}, \cdots, 2^{29}$ and generate a plot `task2.pdf` which plots the time taken by your algorithm as a function of `n` when `threads_per_block` = 1024 and `R` = 128. Overlay another plot which plots the same relationship with a different `threads_per_block` of your choice.

(d) Going beyond the call of duty, do if you wish to: Compare the scaling results with the results obtained in a previous assignment where you did a similar scaling analysis using a sequential implementation on the CPU. If you do this, drop your plots on Piazza. Please comment on the noted results.