CSCE 4930: GPU Computing
Homework 2

This assignment (and subsequent ones) should be completed by teams of 2 to 3 students. Teams of 2 students are generally preferred. Teams of 3 might be required to implement extra functionalities. Only one submission per team is required.

You are required to implement and run all the tasks below on a computer having a CUDA-Enabled device. You can use any such computer you have access to, or you can use Google Colab. Please specify the specific GPU you are using and its CUDA compute capability in your assignment submission. Your submission should be a single compressed folder (e.g., zip) containing a source code subfolder for each task and a single PDF including output screenshots and question answers for all tasks.

For all tasks you are required to do the following:

1. Write a base (sequential) function that implements the required functionality.
2. Write a CUDA-kernel and a wrapper function that initializes the necessary device data and calls that kernel to implements the same functionality. **Make sure to use error checking for all CUDA API calls**.
3. Write a full program that calls both functions on the same input data and verifies that they both produce the same output.
4. Time both versions and compute the speedup (or slowdown) obtained from parallelization. When timing the parallel version, you should do it in two different ways and compute the speedup in each case:
    a. Timing the kernel only
    b. Timing the entire wrapper function (including the memory allocation and data transfer overheads).

Here is ***one*** possible way to time a c function or code segment:

```
#include <time.h>
clock_t start = clock();
/* code you want to time */
clock_t stop = clock();
double time_spent = (double)(stop - start) / CLOCKS_PER_SEC;
```

Please note that launching a kernel is asynchronous with respect to host code. That is, the kernel threads will start executing without blocking the CPU. So, the next few statements following the kernel call (which are statements to be executed by the host) will be executed as a CPU thread running in parallel with the GPU kernel threads. This means that if you want to time the kernel only, you should force the CPU code to wait until the kernel has finished execution using the cudaDeviceSynchronize() function, so the timing code should look like this:

```
clock_t start = clock();
f1_kernel<<<…,…>>>(…,…,…);
cudaDeviceSynchronize();
clock_t stop = clock();
double time_spent = (double)(stop - start) / CLOCKS_PER_SEC;
```

Task 1:

Write a program to perform matrix multiplication of any two floating-point matrices A and B. A and B do not need to be square matrices; however, their dimensions should be compatible. i.e, the number of columns of A has to be equal to the number of rows of B. A and B should be initialized to random values by the host, but their dimensions should be specified in the program using **`#define`** directives. Your parallel implementation should use multiple 2D blocks of threads but no shared memory. You program should print the performance of the sequential and parallel versions in GFLOPS. Provide results for each of the following block sizes: 16x16, and 32x32 **(Teams of 3**

should also report on block sizes of 4x4 and 8x8). **What is the GPU utilization in each case?** Note: for testing purposes, you try relatively small values for the dimensions of the matrices, but to provide timing results, A and B matrices have to be large enough to make parallelism useful (try values larger than 1000 for each dimension).

Task 2:

Repeat Task 1 using shared memory (and tiling) for the parallel implementation.

Task 3:

Repeat Task 2 with thread granularities of 2 and 4. i.e, each thread block will compute 2 or 4 elements in the output matrix (Each block of threads will be responsible for computing 2 or 4 output tiles). **Teams of 3 should also report on thread granularity of 8**.

- Submission Deadline: Sunday, November 6, 2022 11:59 PM.
- Submission method: Electronically through BB.
- Your submission should be **a single compressed folder (e.g., zip)** containing a single PDF report including output screenshots from all tasks and answers to all questions. The compressed folder should also contain a source code subfolder for each task.
- Check the readability of your submission. Hard to read submissions will be ignored.
- Plagiarism is NOT tolerated and will be reported.
- Late submissions will be penalized as described in the course syllabus and will NOT be accepted at all if more than 24 hours late.