ME 759
High Performance Computing for Engineering Applications
Assignment 6
Due Friday 10/21/2022 at 9:00 PM

Submit responses to all tasks which don't specify a file name to Canvas in a file called assignment6.{txt, docx, pdf, rtf, odt} (choose one of the formats). Submit all plots (if any) on Canvas. Do not zip your Canvas submission.

All *source files* should be submitted in the `HW06` subdirectory on the `main` branch of your homework GitLab repo with no subdirectories. For this assignment, your `HW06` folder should contain `task1.cu`, `mmul.cu`, `task2.cu`, `scan.cu`.

All commands or code must work on *Euler* with only the `nvidia/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 `Assignments/HW06` directory of the ME759 Resource Repo. Do not change any of the provided files since these files will be overwritten with clean, reference copies when grading.

---

1. (40 pts) Linear algebra is ubiquitous in many applications. BLAS (Basic Linear Algebra Subprograms) libraries implement a myriad of common linear algebra operations and are optimized for high performance. Some of these libraries target HPC hardware. We will use cuBLAS, which targets Nvidia GPUs. See here for the documentation.

   a) BLAS libraries group functions into three levels. What do all Level 1 functions have in common? Level 2? Level 3? In other words, how did they decide how to group these functions?

   b) Some functions are specialized for performing their operations when the structure of the input matrix or vector is known. List and briefly explain two such functions which assume something about their input structure in order to optimize the computation.

   c) Implement the `mmul` function as declared and described in `mmul.h` in a file called `mmul.cu`. You should use a single call to the cuBLAS library to perform the entire matrix-matrix multiplication (gemm).

   d) Write a test file `task1.cu` which does the following:
      - Creates three `n`×`n` matrices, `A`, `B`, and `C`, stored in **column-major** order in **managed memory** with random `float` numbers in the range [-1, 1], where `n` is the first command line argument as below.
      - Calls your `mmul` function `n_tests` times, where `n_tests` is the second command line argument as below.
      - Prints the **average** time taken by a single call to `mmul` in *milliseconds* using CUDA events.
      - Compile: `nvcc task1.cu mmul.cu -Xcompiler -O3 -Xcompiler -Wall -Xptxas -O3 -lcublas -std c++17 -o task1`
      - Run (where `n` is a positive integer): `./task1 n n_tests`
      - Example expected output:
        `11.0`

   e) On an Euler compute node, run `task1` for each value $n = 2^5, 2^6, \cdots, 2^{14}$ and generate a plot (`task1.pdf`) of the time taken by `mmul` as a function of `n`. You should decide `n_tests` by yourself. It should not be too small so the timing is less accurate, or too large so it takes a long time to run.

   f) Comment briefly on how your tiled matrix multiplication code in HW05 works compared to this cuBLAS-based implementation in terms of efficiency (for problem sizes up to covered by the tests you have done with both implementations). If you dropped HW05, you can refer to the scaling plots reported by your peers on Piazza.

2. (60 pts)

   a) Implement in a file called `scan.cu` the function `scan` as declared and described in `scan.cuh`. Your `scan` should call a kernel function `hillis_steele`, which you will implement to conduct an **inclusive scan** with the Hillis-Steele algorithm given in Lecture 15. `scan` may also call other kernel functions that you write in `scan.cu`. *None* of the work should be done on host, only in the kernel calls. Note that it is important that your `scan` is able to handle general values of `n` (for example, values which are not multiples of 32 (or your block size)). You have some freedom when writing the `hillis_steele` kernel to add a small amount of work that may help complete the scan. You may also allocate some additional memory (less than the size of the input array) in the `scan` function to help complete the function. Feel free to use any code that was provided in the ME759 slides, if at all useful.

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

      - Create and fill an array of length `n` with random `float` numbers in the range [-1, 1] using **managed memory**, where `n` is the first command line argument as below.
      - Call your `scan` function to fill another array with the results of the inclusive scan.
      - Print the last element of the array containing the output of the inclusive scan operation.
      - Print the time taken to run the full `scan` function in *milliseconds* using CUDA events.
      - Compile: `nvcc task2.cu scan.cu -Xcompiler -O3 -Xcompiler -Wall -Xptxas -O3 -std c++17 -o task2`
      - Run (where `n` is a positive integer, `n`≤`threads_per_block`*`threads_per_block`): `./task2 n threads_per_block`
      - Exampled expected output:
        `1065.3`
        `1.12`

   c) Include in your Canvas file, the output of `cuda-memcheck ./task2 n threads_per_block`, where `n` $= 2^{10}$ and `threads_per_block` $= 1024$[1].

   d) On an *Euler compute node*, run `task2` for each value `n` $= 2^{10}, 2^{11}, \cdots, 2^{20}$ with `threads_per_block` as 1024 and generate a plot `task2.pdf` which plots the time taken by your algorithm as a function of `n`.

   e) (Challenge problem; done for glory)
      Solve this problem by having the host launch only one kernel call, and with no data processing done on the host. Hint (not necessarily useful, it depends how you go about this solution): You can have a CUDA thread launch kernels itself.

[1] this helps you and the graders make sure your code handles memory correctly. And if your code does, `cuda-memcheck` will not change the expected output format significantly, apart from adding a few lines indicating no error found. `cuda-memcheck` does make your code slower, so do it when debugging, do not do it when acquiring timing data.