Major Project: Parallelizing Cholesky Factorization Algorithm on a GPU

A symmetric positive definite[*] (SPD) matrix A can be factored into an upper triangular matrix[†] R and its transpose R^T that allows easy solution of a linear system of the form Ax = b. Since A = R^TR, one can compute x by solving two triangular systems: $R^Ty = b$ to determine y, followed by Rx = y to obtain x. Solving triangular systems is straightforward and has a complexity of $O(n^2)$ for an nxn matrix, which is proportional to the number of elements in the matrix.

Cholesky factorization can be viewed as computing R for a matrix A via a process similar to Gaussian elimination. This project requires you to develop a parallel implementation of the Cholesky factorization algorithm on a GPU and to study its performance. You are provided a CUDA-based starter code that computes the factorization on the GPU using a single thread.

The starter code computes the factor R for a matrix A via the Cholesky factorization algorithm. The code initializes the matrix `A` on the host, ensuring that it is SPD, and copies it to the device as `dA`. The code includes a device routine called `device_cholesky_factorization` that transforms the device copy `dA` into the upper triangular matrix R, overwriting `dA` in the process with entries of R. The updated `dA` is copied back to the host as `R` and The code also confirms that the entries of R^TR are within tolerance of `A`. The code also provides guidance on use of CUDA resources for error detection and timing.

The primary intent of the starter code is to help you get started on GPU program development quickly so that you are able to focus on the parallelization and optimization aspects of the project.

1. (75 points) Develop a parallel implementation of the Cholesky factorization algorithm that runs on the GPU. Your code should correctly compute R using multiple GPU threads and should demonstrate speedup over the single-thread code. 10 points out of 75 are reserved for implementations that use more than one multiprocessor on the GPU to compute the factorization (recall that one needs to create more than one block of threads in the grid in order to use more than one multiprocessor).
2. (25 points) Develop a report that describes the parallel performance of your code and its dependence on the grid structure used for kernel invocation. This will require you to conduct a variety of experiments with varying matrix sizes and grid structures to understand the behavior of your program on the GPU. Discuss any design choices you made to improve the parallel performance of the code and how they relate to actual observations. Include any insights you obtained while working on this project. Lastly, include a brief description of how to compile and execute the code.

[*] A symmetric positive definite (SPD) matrix is a matrix that has positive real eigenvalues, and permits factorization of the for $A = R^TR$, where R is a real valued upper triangular matrix and R^T is its transpose.
[†] An upper triangular matrix is one with zero entries below the diagonal.

Submission:

Upload two files to Canvas:

1. A **single zip file** consisting of the code you developed.
2. The project report as a single PDF or MSWord.

Helpful Information:

1. You may use Grace or Faster for this assignment.
2. Information on compiling and running CUDA programs on a GPU for Grace is available at https://hprc.tamu.edu/kb/User-Guides/Grace/Compile/#cuda-programming.
3. To develop code interactively, log on to one of the GPU-equipped login nodes on Grace; these nodes are named graceX.hprc.tamu.edu, where X is to be replaced by the numbers 1, 2, or 3. See https://hprc.tamu.edu/kb/User-Guides/Grace/Hardware/#login-nodes for additional information. If you use portal.hprc.tamu.edu to access Grace, you are randomly assigned a node where X can be 1-5. You will then need to use ssh to log into a GPU-equipped login node from the initial shell.
4. Load the modules for compiler and CUDA prior to compiling your program. At the time of preparing this assignment, the following module combination worked on Grace (later modules did not work):
   ```
   module load intel/2023a CUDA/12.2
   ```
5. Compile C/C++ programs using `nvcc`. For example, to compile `code.cu` to create the executable `code.exe`, use
   ```
   nvcc -o code.exe code.cu
   ```
6. The run time of a code should be measured when it is executed in dedicated mode. Check out https://hprc.tamu.edu/kb/User-Guides/Grace/Batch/#job-examples for sample batch files. To execute the code on a GPU-equipped node, you must use the submission options for GPUs.