Tiled Matrix Multiplication

Due Date: Friday, Feb. 3 at 11:59 pm

Objective

The purpose of this lab is to implement a tiled dense matrix multiplication routine using shared memory.

Prerequisites

Before starting this lab, make sure that:

- You have completed the “Naive Matrix Multiplication” MP

Instruction

Edit the code in the “Code” tab to perform the following:

- allocate device memory
- copy host memory to device
- initialize thread block and kernel grid dimensions
- invoke CUDA kernel
- copy results from device to host
- deallocate device memory
- implement the matrix-matrix multiplication routine using shared memory and tiling

Instructions about where to place each part of the code is demarcated by the `//@@` comment lines.

Suggestions (for all labs)

- Backup your code regularly.
- Do not modify the template code provided -- only insert code where

the `//@@` demarcation is placed

- Develop your solution incrementally and test each version thoroughly before moving on to the next version
- Do not wait until the last minute to attempt the lab.
- If you get stuck with boundary conditions, grab a pen and paper. It is much easier to figure out the boundary conditions there.
- Implement the serial CPU version first, this will give you an understanding of the loops
- Get the first dataset working first. The datasets are ordered so the first one is the easiest to handle
- Make sure that your algorithm handles non-regular dimensional inputs (not square or multiples of 2). The slides may present the algorithm with nice inputs, since it minimizes the conditions. The datasets reflect different sizes of input that you are expected to handle
- Make sure that you test your program using all the datasets provided (the datasets can be selected using the dropdown next to the submission button)
- Check for errors: for example, when developing CUDA code, one can check for if the function call succeeded and print an error if not via the following macro:

```
#define wbCheck(stmt) do { \ cudaError_t err = stmt; \ if (err != cudaSuccess) { \ wbLog(ERROR, "Failed to run stmt ", #stmt); \ wbLog(ERROR, "Got CUDA error ... ", cudaGetErrorString(err)); \ return -1; \ } \ } while(0)
```

An example usage is `wbCheck(cudaMalloc(...))`.

Plagiarism

Plagiarism will not be tolerated. The first offense will result in the two parties getting a 0 for the machine problem. Second offense results in a 0 for the course.