3460:677 LAB 4: MATRIX MULTIPLICATION

Objective

In this lab the student will implement dense matrix multiplication, first without optimizations such as tiling and usage of shared memory, then with those optimizations.

Prerequisites

Before starting this lab, make sure that:

- You have completed the "Vector Addition" Lab.
- You have completed Online Modules 4A and 4B.

Local Setup Instructions

The most recent version of source code for this lab can be found in a `.zip` file on the class web page. As before, download it to your Windows account on a lab machine, then copy it to your `tesla` account. Log into `tesla` and `unzip` the files there. (Note that this has been designed to work on and tested on `tesla`. You may do your work on other CUDA-capable equipment but you are on your own altering the seed files so that they execute.)

Overview

The **`Lab4`** directory contains six files:

- **`run.sh`** is a Linux batch file that compiles and executes a named CUDA program;
- **`basicMult.cu`** is the code template for the un-optimized dense matrix multiplication;
- **`tiledMult.cu`** is the code template for the optimized dense matrix multiplication; and
- **`input0.raw`** and **`input1.raw`** are input data files to test your program, with **`output.raw`** the correct answer used to check your work.

The code templates are provided as a starting point. The code handles the import and export of data files as well as the checking of the solution. You are expected to insert your code in the sections demarcated with `//@@`. Leave the other code unchanged. To compile and execute use the commands **`./run.sh basicMult`** and **`./run.sh tiledMult`**.

Instructions

Edit the code in both **`basicMult.cu`** and **`tiledMult.cu`** 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

Additionally:

- In **`basicMult.cu`**, remember to use **`cudaMemset`** to zero out the result matrix on the device before launching the kernel.
- In **`tiledMult.cu`**, implement the matrix-matrix multiplication routine using shared memory and tiling. Experiment with tile widths of 4, 8 and 16. Compare timing information for these three experiments and draw conclusions as part of your lab report.

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

Lab Report and Submission

When finished write a one-to-two page lab report (single-spaced, 1-inch margins, 12- to 14-point font size) summarizing your work and findings. In your report answer these questions:

(1) How many floating operations are being performed in each of your matrix multiply kernels? Explain.
(2) How many global memory reads are being performed by each of your kernels? Explain.
(3) How many global memory writes are being performed by each of your kernels? Explain.
(4) Describe what possible further optimizations can be implemented to each of your kernels to achieve performance speedups.
(5) Name three applications of matrix multiplication.
(5) Compare the implementation difficulty of the **`tiledMult.cu`** kernel to that of the **`basicMult.cu`** kernel. What difficulties did you have with this implementation?
(6) Suppose you have matrices with dimensions bigger than the max thread dimensions. Sketch an algorithm that would perform matrix multiplication algorithm that would perform the multiplication in this case.
(7) Suppose you have matrices that would not fit in global memory. Sketch an algorithm that would perform matrix multiplication algorithm that would perform the multiplication out of place.

Submit the two edited `.cu` files and your lab report by the due date for grade.