Naive Matrix Multiplication

Due date: April 21 at 11:59 pm

Objective

The purpose of this lab is to implement a basic dense matrix multiplication routine.

Location of Assignment Starter Code and Required Background Code :
`/work1/sadasivan/csep590_class_root/csep590_class/assignments`

Prerequisites

Before starting this lab, make sure that:

- You have completed the “Vector Addition” MP

Instruction

Edit the code in the ‘Code’ tab to perform the following:

- allocate device memory
- copy host memory to the device
- initialize thread block and kernel grid dimensions
- invoke HIP kernel
- copy results from device to host
- deallocate device memory

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

Suggestions (for all labs)

- Back up 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 HIP code, one can check for if the function call succeeded and print an error if not via the following macro:

```
#define wbCheck(stmt) do { \
      hipError_t err = stmt; \
      if (err != hipSuccess) { \
            wbLog(ERROR, "Failed to run stmt ", #stmt); \
        wbLog(ERROR, "Got HIP error ... ", hipGetErrorString(err)); \
            return -1; \
      } \
} while(0)
```

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

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.