ECS 154B Lab 5, Spring 2018
Due by 11:59 PM on June 3, 2018
Via Canvas

Objectives

- Experiment with running code on GPUs.
- See the differences in both code and runtime for programs written for CPUs versus GPUs.
- Familiarize yourself with NVIDIA's CUDA compute platform.
- Gain insight into how GPUs work.

Description

This assignment was ~~shamelessly stolen~~ *adapted* from an assignment by UC Davis' John Owens in his EEC 289Q class from Winter Quarter 2018.

Graphics processing units (GPUs) have a wide variety of uses. Their most common and well known use is for graphics applications, especially for PC gaming. They are also especially good for artificial intelligence, machine learning, and cryptocurrency workloads, which resulted in a mass increase of demand (and price) for GPUs in late 2017.

One lesser-known use of GPUs is in the scientific computing environment. Some of the newest supercomputers that are being built for Oak Ridge and Lawrence Livermore National Laboratories are being built using GPUs. It is that use that we will explore in this assignment.

CUDA is a parallel computing platform developed by NVIDIA for their GPUs. It was designed to make writing parallel programs for GPUs easier. It is not the only platform that runs on GPUs, however. Both NVIDIA and AMD GPUs support OpenCL, the free alternative to CUDA.

Details

You will be going through the "An Even Easier Introduction to CUDA" tutorial on the NVIDIA Developer Blog. After you complete the tutorial, you will implement a separate kernel and compare CPU and GPU runtimes.

Compute Environment

The only requirement for this assignment is that you use one consistent machine (and thus GPU) for the entirety of the tutorial. If you would like to use your own machine that has an NVIDIA GPU, feel free. You will need to download the CUDA toolkit on your own. However, I will not provide help for toolchain issues on non-CSIF machines. If you don't have an NVIDIA GPU, or don't want to install the CUDA toolkit, you will need to use the CSIF.

The CSIF machines `pc21` through `pc60` have NVIDIA GeForce GTX 960s in them. You can do the entire assignment by SSHing in - there is no need to go to the labs if you don't want to.

Note that the commands `nvcc` and `nvprof` are not in your default path on the CSIF machines. The commands are located at `/usr/local/cuda-8.0/bin/nvcc` and `/usr/local/cuda-8.0/bin/nvprof`, respectively. It is recommended that you create shell scripts, aliases in your `.bashrc` file, or add these to your path, to make it easier on yourself.

Assignment

1. Tutorial

The file to submit for this part is `add_grid.cu`, as listed in the tutorial.

Go through the tutorial, up to the *Summing Up* section. You do not need to do anything in the *Exercises* section. There are three different files created during the course of the tutorial: `add.cu`, `add_block.cu`, and `add_grid.cu`. Only turn the last file in.

Some of the important terms mentioned in the tutorial (such as threads, blocks, and grids) may make their way into interactive grading. It is suggested that you read through the tutorial carefully.

2. `daxpy`

The file to submit for this part is your modified version of `part2.cu`. The original `part2.cu` is on Canvas.

Now that you have a little bit of experience writing CUDA code, it's time to implement something on your own. Implement the GPU kernel `daxpyGPU()` at the top of `part2.cu`. `daxpy` takes in an array size and four input floats: `a`, `x`, `y`, and `result`. Then, it performs the following calculation:

$$\text{result} = a \cdot x + y$$

At the end, the code compares the CPU and GPU versions, and calculates the maximum error. This value printed must be zero at the end of the program.

You can view the CPU version of the code in the function `daxpyCPU()` right below `daxpyGPU()`. Your GPU code must be parallelized, using proper CUDA structures. You may not have one GPU thread do the entire calculation. If you need help on parallelizing the code, look back at the tutorial from part 1.

After you have a working version of `daxpyGPU()`, compare runtimes, and leave a comment at the top of `daxpyGPU()` with the two runtimes. The first paragraph of the results of `nvprof` show how long it took the GPU to run the kernel. How do the results compare for the two functions? The GPU should be significantly faster than the CPU for the calculation. If it is not, you most likely made a mistake in your parallelization, or you're looking at the wrong section of the results.

Feel free to play around with some of the parameters, like the block size `#define` at the top of the file, or the `arraySize` variable. Those values can significantly change the runtime for both functions. However, make sure that the file you turn in only changes the contents of `daxpyGPU()`.

Grading

- 25% for a working `add_grid.cu`.
- 25% for a working `part2.cu` that implements a parallelized version of `daxpyGPU()` correctly, with the two runtimes at the top of the function. No modifications should be made to the rest of the file.
- 50% for interactive grading. Interactive grading may ask questions about material from the tutorial. Make sure you familiarize yourself with the important terms and things you modified during the course of the tutorial, such as threads, blocks, and grids.
- Partial credit at the grader's discretion.

Submission

Read the submission instructions carefully. Failure to adhere to the instructions will result in a loss of points.

- Upload your versions of `add_grid.cu` and `part2.cu`.
- In the text submission box, include the following:
  - The names of you and your partner.

  - Any difficulties you had.
  - Anything that doesn't work correctly and why.
  - Anything you feel that the grader should know.

- Only one partner should submit the assignment.
- You may submit your assignment as many times as you want.

Late Policy

You may submit up to 48 hours late for partial credit. The partial credit that you receive is determined by the following equation:

$$\text{final grade} = \text{original grade} \cdot \left(1 - \frac{\text{hours late}^2}{48^2}\right)$$

This system penalizes you less if you submit closer to the original deadline, compared to a straight linear drop-off. Note that if you do not attend interactive grading, you will receive a 0 regardless.