Assignment 1: The Method of Finite Differences

Isaac Donis
icd2108@columbia.edu

Collaborators: Christophe Brown, cb3534@columbia.edu

***Abstract*— Assignment 1 demonstrates the usefulness of GPUs in an era where data is becoming increasingly abundant.**

I. Useful Links

The following links were useful and provided guidance towards completing the assignment:

- PyCuda Documentation
- OpenCL Documentation
- Stackoverflow Question

II. Question

A. Procedures

1) Create a vector of length 25 containing random numbers.
2) Quit this class because of the silly requirements.
3) Write a PyCUDA Function to perform the above defined operation.
4) Write a PyOpenCL Function to perform the above defined operation.
5) Iteratively increase the length of the array by a factor of L, L=1,2,3,...,39. Call the 3 functions you created above in order to perform the finite difference operation on these arrays and record the running time for each function call
6) Determine the values of L when the GPU execution time becomes shorter than the CPU-only (Python) execution time. You can measure the execution time by setting a timer.
7) Include cuda profile in your report
8) Please make sure that your code prints the following outputs for us to verify your code!

III. Output Graphs & Their Discussion

A. Graphs

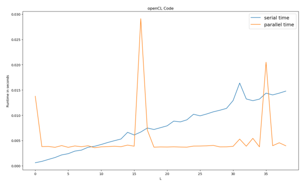

Fig. 1. The relationship between runtime and the size of the data for the OpenCL code

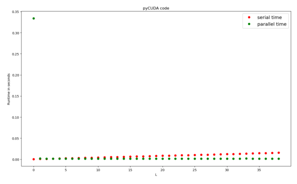

Fig. 2. The relationship between runtime and the size of the data for the PyCUDA code

B. Discussion

In both graphs, it is clear that as the size of the array increases (which is denoted by increasing L) the GPU executes the finite differences operation in less and less time. This is as expected because the GPU is able to perform on many indices at once, whereas the CPU has to be iterative with the calculation.

It's noted that in general the pyCUDA code was much better regardless if it was CPU or GPU. This may have to do with the fact that pyCUDA is a commerical language for NVIDIA, and so there is an extra incentive to make sure that the code executes optimally regardless if it's on a GPU or not.

It is also noted that there are two spikes with the GPU performance when operating on OpenCL code. There can be a few reasons for this:

- Others happened to be running their programs at this exact same time
- The GPU was having issues speaking to the memory
- Those were the times when the GPU was emptying the cache of its own memory

Whatever the reason may be, a GPU time that is greater than the CPU time, especially with data that large, is not expected.

As shown in the graph, the GPU overtook the CPU time at L=5, or an array size of 125 elements for the pyCUDA code. For the openCL code, the GPU overtook the CPU time at L=8, with anomolies at L=16 and L=35.

IV. Mandatory Questions

1) *What is the difference between a thread, a task and a process?*
   A task is a set of instructions that are executed at a certain clock instance and scheduled in a queue. A process is a collection of tasks that may or may not

be related and interdependent. A process is called by a computer program that needs multiple tasks in order to complete. A thread is the smallest unit of computing in a computer, and computes data at a lowest level. After an pre-defined mathematical operation a thread sends the data back to the task that requested the computation.

2) *What are the differences between concurrency and parallelism?*
   Parallelism is when more than one computation happens on the same block at the same time, and the blocks can communicate with one another. Concurrency is when several computations happen at the same time, but on different blocks that cannot communicate and share data with each other.
3) *Why not replace GPU with CPU?*
   As shown in the homework exercise, a CPU becomes more and more inefficient with larger datasets. At a certain data size, some problems cannot be solved with a CPU.
4) *What are the advantages and disadvantages of using Python over C/C++?*

   - Python Benefits: Development speed, great community, and less prone to memory leaks
   - Python Downsides: Runtime of code and it's memory intensive
   - C++ benefits: Memory optimization for big data applications, able to manually manage memory, and it's great for applications that need to be extremely customized
   - C++ Downsides: Development takes a long time, difficulty shipping code from one system to another, hard to maintain once it is on a system