ECE 459: Programming for Performance
Assignment 3

Jeff Zarnett & Patrick Lam

February 24, 2018 (Due: March 12, 2018)

In this assignment, you will convert your code that performs the Coulomb's Law simulation from assignment 2 to use OpenCL. This problem is a great candidate for OpenCL because computation of the force on each point p at a given time is independent of the computation of any other point at that same time.

OpenCL is covered in the lecture notes, but you can read Andrew Cooke's notes; they are similar to the OpenCL lecture, but they may make more sense to you[1].

Using OpenCL on `ecetesla0` should be fairly straightforward as long as you have the right compiler options. The Makefile I've provided should work for you. You are also free to use OpenCL on your own computer, if you choose. There are C++ as well as C bindings for OpenCL but you should use **only the C++ bindings**. The C binding has a nasty habit of crashing the machine (and/or rendering OpenCL unusable). If you use the C bindings, you will get 0 marks and we will be slightly displeased.

Getting started. Check `git.uwaterloo.ca` for your repository which should be automatically created for you and populated with the starter files. The starter files are just the Makefile and some OpenCL boilerplate that will be necessary. Plus also some test cases.

Then you can copy in your source files from assignment 2, question 3. You'll need to modify them extensively but it's probably much better to start with those than to start from scratch. But it's up to you!

What to submit. Push C++ files as well as your kernel file(s), containing the OpenCL version of your code, along with a brief report (max 1 page).

Coulomb's Law Problem: Again!

For a recap of the algorithm is supposed to work, please refer to the assignment 2 PDF (repeating it here only introduces the possibility of inconsistency or error). The input file format and the argument order will be the same at the command line. The arguments mean the same thing as they did in assignment2. The compiled executable file is named slightly differently to note that it's OpenCL.

The output file produced by the OpenCL version should be exactly the same as an output file produced by the OpenMP or sequential version for the same input file.

The goal of the assignment is to correctly convert the CPU code from assignment 2 into GPU (OpenCL) code. Correctness counts the most, but efficiency is also a good thing (you don't want it to be massively slower). It might now be clear why we used `float` in the previous assignment instead of `double`: because GPUs are very, very good at working with `floats`.

[1] `http://www.acooke.org/cute/APractical0.html`

Converting Your Code to OpenCL

Step one would be to remove all OpenMP annotations as your code should not contain OpenMP directives (the goal is, after all, to do work in OpenCL).

The OpenCL kernel is written in a language that is very C-like but it does have some significant limitations compared to C. This makes it difficult to use structures like the `Vec3` that you were provided in assignment 2. But rest assured that we didn't trick you into doing something that would force a massive rewrite later. OpenCL gives you the types `float3` and `float4` (and other variants like 2, 8, 16...), which can take the place of `Vec3`.

A `float4`, for example, is a grouping of 4 single-precision floating point numbers. It is equivalent to a `struct` composed of 4 `float` variables. It has properties (like `.x`) to access the components, and there are already defined arithmetic and comparison operations on them. So you should be able to drop these types into your program, replacing `Vec3`, without great difficulty. The links below should be helpful in getting familiar with the vector types and how to use them. They are both from the same article; the whole article might be worth a read but you should definitely read at least these two pages to understand the types and how the operations on them work:

`http://www.informit.com/articles/article.aspx?p=1732873&seqNum=3`

`http://www.informit.com/articles/article.aspx?p=1732873&seqNum=10`

Beware, of course, of mismatches such as writing `float` when you mean `float4` or similar. Note that the types have slightly different names depending on where you use them: in the OpenCL kernel it's called `float4` but in your C++ code it is `cl_float4`, for example.

Once you have converted to using the OpenCL vector types, it's probably a good idea to test with some test cases and make sure that nothing was broken by this. Then make a commit to save your work and you'll then go on to the next part, which is handing work over to the GPU. You do set-up and configuration in the host (CPU, C++) code, before giving the work to the GPU and collecting the results back to the host code.

You need to do at least the following operations in GPU (OpenCL kernel). You may **not** do these things in CPU code because it defeats the purpose of the assignment:

1. the y_1 computation
2. the z_1 computation
3. checking for error larger than the maximum allowed.

Hint for item 3: You might think of this as either a reduction, or as finding the max value...

Your solution should probably consist of three kernels (but could be slightly more or less, depending).

Report

In the report, write about your design choices and results (max half a page). Also write about what worked well in this assignment and where you had difficulties, as we could use this in the future to give more guidance (also max half a page). Please use LaTeX for this; a template is provided for you in the base code that you cloned.

Grading

60% of the marks for correctness of conversion; 30% for efficiency; 10% for report. Efficiency in this case means (1) clean code conversion, (2) leverages the GPU well (i.e., the code isn't executed in a very inefficient way), (3) doesn't do unnecessary work. Your code will be evaluated on `ecetesla0` so make sure it works as expected there.