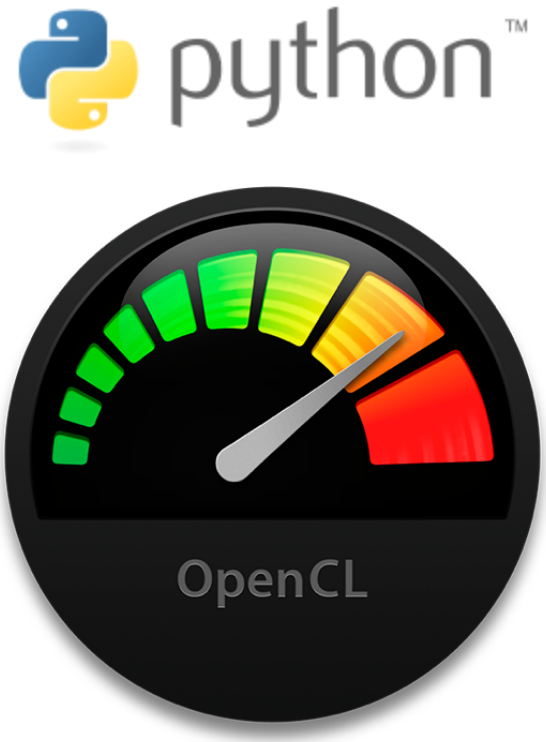
ECEN 5593

Advanced Computer Architecture

Summer 2016



Final Project Proposal

Diana Southard, Yaffe Kravitz

Introduction

When attempting to utilize the benefits of programming on the GPU for high-performance parallel computing, the two major options are CUDA and OpenCL. CUDA has the limitation of only being accessible on NVIDIA graphics cards whilst OpenCL is able to execute on GPUs from a multitude of popular manufacturers (Intel, AMD, NVIDIA, and more). However, OpenCL is a more terse language like C – as such, it has a difficult learning curve for the new programmer unfamiliar with the language.

In order to overcome this difficulty, the features of OpenCL have been wrapped around with the more-friendly Python language to create PyOpenCL. PyOpenCL can be written in a normal Python environment but is able to access all the features of OpenCL, providing shortcuts and enhancements for common tasks. With Python's simpler learning curve, even a programmer unfamiliar with it can soon be executing PyOpenCL code.

Topic

We will be investigating PyOpenCL, specifically the ease of using it and the benefits of utilizing the GPU to speedup complex mathematical operations. So far this semester, we have learned some fundamental CUDA concepts, some of which map over to OpenCL, but have not spent as much time with the OpenCL language itself. This final project will increase our exposure to another GPU programming language. Additionally, it will allow us to explore Python, which in a popular programming language useful for computational scripting.

Methodology

We will measure the time it takes to operate on a set of variables. The operations will be performed first on the CPU using pure Python, and then converted to PyOpenCL to be performed on the GPU. The difficulty of porting code over to the GPU via PyOpenCL will be noted, though this is a purely subjective note as programmers more familiar with the languages will not have as much difficulty as we will.

We will begin by investigating simpler mathematical operations by implementing a vector adding function. After successfully creating the vector addition function, we will then implement a matrix multiplication function for more complex operations. In both cases, the size of the input and the blocksize of the kernel can be adjusted. We will time both functions, comparing the time difference for varying input/block-size lengths and execution times between the CPU and GPU. In addition, we will keep track of any difference between the CPU and CPU results in order to validate our kernel functions.

All measurements will then be exported to an Excel worksheet and graphed for ease of noting trends, changes, and marking speedups. This will also check that there is a minimal difference between the CPU and GPU results.

Our main comparisons will be between PyOpenCL and Python. If there is enough time, we can explore the option of writing the programs in pure OpenCL and comparing any timing differences.

Goal

We believe that we will be able to explore the benefits of parallelization through the use of the GPU to speedup calculations. This should show the limits, as well: for smaller operations, the additional time needed to transfer memory to and from the GPU may negate the speedup in

performing those calculations there, and we should be able to note when operations become large enough to result in an overall speedup when using PyOpenCL vs Python.

We will also be able to report about the ease of using PyOpenCL for implementing these functions. The purpose of the language is to simplify the use of OpenCL. If there are any excessive difficulties in figuring out its usage, we will note it.

Code reference

The following Github Repos shows multiple examples on how to use PyOpenCL, including matrix multiplication:
https://github.com/pyopencl/pyopencl
https://github.com/stefanv/PyOpenCL

Project References and Resources

https://mathema.tician.de/software/pyopencl/
http://enja.org/2011/02/22/adventures-in-pyopencl-part-1-getting-started-with-python/
https://developer.nvidia.com/pyopencl