ECE574: Cluster Computing – Homework 9
OpenCL

Due: Friday 18 April 2025, 5:00pm

1. **Background**

   - In this homework we will take the sobel code from earlier homeworks and parallelize it using OpenCL.

2. **Setup**

   - For this assignment log into the same Haswell-EP machine we used in previous homeworks. As a reminder, use the username handed out in class and ssh in like this
     `ssh -p 2131 username@weaver-lab.eece.maine.edu`
   - Download the code template from the webpage. You can do this directly via
     `wget https://web.eece.maine.edu/~vweaver/classes/ece574/ece574_hw9_code.tar.gz`
     to avoid the hassle of copying it back and forth.
   - Decompress the code
     `tar -xzvf ece574_hw9_code.tar.gz`
   - Run `make` to compile the code.
   - You probably are best using the provided code as a base, but your CUDA kernel codes can be used mostly unchanged as your OpenCL kernels.

3. **Make the convolve and combine codes work with OpenCL (7 points)**

   The provided code does most of the more annoying verbose OpenCL setup for you.

   The primary thing you have to do is convert your CUDA code to be OpenCL kernels.

   (a) Edit the file `sobel_opencl.c`

   (b) Be sure to comment your code!

   (c) Put your kernel code for convolve and combine into the `KernelSource_combine` and `KernelSource_convolve` strings.

      i. Note, these are strings that are one long string.

      ii. Feel free to change the parameters to the kernels, but if you do, you will need to change the command line argument code in `main()`.

      iii. For the square root, you might have to use `sqrt()`, not `sqrtf()`, but still have to force it to use a "float" value using a cast (something like `sqrt((float)value);`.

   (d) As stated, if you change the data structures you use, or the arguments to the kernels, you will need to change the appropriate code that calls the kernels in `main()`

   (e) I've provided the PAPI measurement code. Be sure if you move things around that it still generates proper results.

4. **Debugging**

   - The most common issue with the code in this assignment is going out of bounds (which can cause segfaults and weird errors). Code that worked fine on CUDA might break here, as CUDA is a bit less strict about out of bounds errors.
   - One way to debug things is to just have your boundary detection code, and set the output to 0 if out of bounds and 0xff if in bounds. If this generates a nice white image with a black edge, you are good. If not, you'll need to find out why.

5. **Performance Measurements (3 points)**

   With OpenCL you can target the code to multiple types of hardware with the same code base. On haswell-ep we have three different OpenCL backends installed. You can use the `clinfo` program to see details on each.

   I have set up the provided code so that it takes a second command line argument (after the filename) to select the OpenCL backend to use. Use this when doing the below measurements.

   (a) First run on the NVIDIA CUDA OpenCL backend, by running things like:
   `./sobel_fine space_station_hires.jpg 0`
   Report the PAPI timing results.

   (b) Next run on the Intel CPU backend (this is a binary only driver from Intel optimized for Intel CPUs):
   `./sobel_fine space_station_hires.jpg 1`
   Report the PAPI timing results.

   (c) Run on the POCL backend (this is an open-source generic backend that uses LLVM to make portable OpenCL code output. In our case it also runs on the CPU)
   `./sobel_fine space_station_hires.jpg 2`
   Report the PAPI timing results.

   (d) Finally, compare the results to the ones you got with your CUDA code from the last homework.

   (e) Question: Which ran fastest? Why?

6. **Submitting your work.**

   - Be sure to edit the README to include your name, as well as the timing results, and any notes you want to add about your something cool.
   - Run `make submit` and it should create a file called `hw09_submit.tar.gz`. E-mail this file to me.
   - e-mail the file to me by the homework deadline.