ECE574: Cluster Computing – Homework 8
CUDA

Due: Thursday 4 April 2019, 11:00am

1. **Background**
   - In this homework we will take the sobel code from earlier homeworks and parallelize it using CUDA.

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 http://web.eece.maine.edu/~vweaver/classes/ece574/ece574_hw8_code.tar.gz`
     to avoid the hassle of copying it back and forth.
   - Decompress the code
     `tar -xzvf ece574_hw8_code.tar.gz`
   - Run `make` to compile the code.
   - You might want to start with the provided code as the GPU code is going to be a lot different than any of the previous implementations. You may use your old code from a previous assignment if you want.

3. **Move "combine" to the GPU (5 points)**

   We will first convert the "combine" routine to run on the GPU.

   (a) Edit the file `sobel_coarse.cu`

   (b) Be sure to comment your code!

   (c) You can implement this any way that works, but what follows is a suggested first implementation:

      i. Use `cudaMalloc()` to allocate memory on the device (GPU) for sobelx, sobely, and the output.

      ii. Copy the host `sobel_x.pixels` and `sobel_y.pixels` to the device using `cudaMemcpy()` (be sure to get the direction right).

      iii. Call the GPU combine code.
      NOTE: the image is too big to simply do a call like the following as in general you are limited to running 256 threads at a time.

```
int image_size=image.x*image.y*image.depth;
cuda_combine<<<1,image_size>>>(image_size,dev_sobelx,
                    dev_sobely,dev_new);
```

      You will need to split the calculations up across a number of blocks of threads.

```
int image_size=image.x*image.y*image.depth;
cuda_combine<<<(image_size+255)/256,256>>>(image_size,dev_sobelx,
                    dev_sobely,dev_new);
```

Your combine function will need to figure out which pixel its operating on ("i" in the sample code) with something like

```
int i=blockIdx.x*blockDim.x+threadIdx.x;
```

where `blockIdx.x` is which block you are in, `blockDim.x` is the number of threads per block, and `threadIdx.x` is your thread offset inside the block.

iv. Copy the results back into `new_image.pixels` using `cudaMemcpy()` (be sure to get the direction right)

v. Add timing calls to PAPI so you can print the following values

A. Time taken to load the jpeg image
B. Time taken to convolve
C. Time to copy the image data to the GPU
D. Time taken to combine
E. Time to copy the image back from the GPU
F. Time to store the jpeg image to disk
G. Total overall time

vi. Some hints on things to try if it's not working:

A. To debug that your kernel works, you can have your cuda_combine routine simply set the output to all 0xff and verify you get an all-white image back.
B. If that works, you can make the output just be a copy of the sobel_x input and make sure you get back what you passed in.
C. When you call `sqrt()` inside the kernel, you might need to cast the value to double before taking the sqrt, otherwise CUDA might complain about you trying to use a host version of the function.
D. nvcc uses C++ to compile things, so be sure you aren't using C++ reserved words (such as "new") as variable names

vii. Note, I don't have slurm configured to handle GPU jobs, so just run the program normally without slurm.

viii. Run on the `space_station_hires.jpg` input

ix. Report the PAPI times reported as your results in the README.

4. **Fine Grained (5 points)**

(a) Modify the code so that the convolves are done on the GPU.

(b) First copy your code to `sobel_fine.cu` and edit it.

(c) Here are some hints. You don't have to do it this way, but this might help you get started.

i. The hardest part here is splitting up the loops into the grid/block/thread paradigm. It is best if you can collapse things into one gigantic loop rather than nested loops. You might want to try to do this in C before attempting it in CUDA.

ii. Remember to skip the edges. If your index variable is `i` and a completely collapsed loop, this means to skip $i < xsize * depth$ at the beginning, $i > xsize * depth$ at the end, and then the $i\%(xsize * depth) < 3$ and $i\%(xsize * depth) > (xsize * depth - 4)$

iii. Before calling your CUDA generic convolve, you will need to upload the appropriate sobelx or sobely matrix to the GPU. This can just be done as an array of 9 integers.

iv. For each point "i" add in the 9 scaled values.

v. Remember that with the collapsed loop there are three separate RGB colors, so instead of indexing the input data like:

```
sum+=in[i-1]*matrix[3];
sum+=in[i]*matrix[4];
sum+=in[i+1]*matrix[5];
```

it will be more like:

```
sum+=in[i-3]*matrix[3];
sum+=in[i]*matrix[4];
sum+=in[i+3]*matrix[5];
```

vi. When debugging it might be helpful to output the sobel_x output and run on the butterfinger input and get that to match exactly before running with both sobel_y and combine hooked up.

(d) Run on the `space_station_hires.jpg` input

(e) Report the PAPI times reported as your results.
How does the total time compare to your fastest CPU-based Haswell-ep run (probably your OpenMP results from homework 5)?

5. **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 `hw08_submit.tar.gz`. E-mail this file to me.
- e-mail the file to me by the homework deadline.