ECE574: Cluster Computing – Homework 8
CUDA

Due: Friday 7 April 2023, 5:00pm

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 https://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 `sobel_coarse.cu` code as the GPU code is going to be different from the previous assignments. However, you may use your old code from a previous assignment if you want.
3. **Make `combine()` use the GPU (7 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 our images have more pixels than the maximum GPU thread count.

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

      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_image);
```

Your combine function will need to figure out which pixel it's 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 using 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. If you try to use plain `sqrt()` in your kernel you might get an error from CUDA. Use `sqrtf()` to force the single-precision square root (which is also much faster than the double-precision version when run on a GPU)

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 The expected output image has an md5sum of `7a17b02fe7e4e676b575f6f66ba4fa01`

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

4. **Run the Convolutions on the GPU (3 points)**

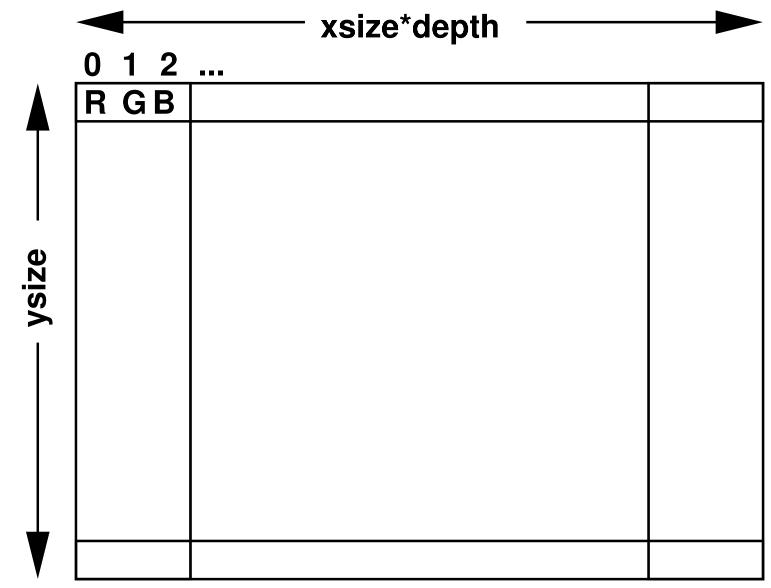
(a) Modify the code so that the convolutions are also done on the GPU.

(b) First copy your `sobel_coarse.cu` over top of `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 collapsing the loops into just a single loop if you haven't already (In theory it is possible to have CUDA operate on a 3D array, but you'd probably have to completely re-do how the data is allocated).

You might want to try doing this in plain C and testing it before attempting it in CUDA.



ii. The next really tricky part is skipping the edges. If your index variable is `i` and a completely collapsed loop, this means skipping the top and bottom rows (skip if $i < xsize * depth$ for the top or if $i > (ysize - 1) * xsize * depth$ at the bottom).
You will also need to not calculate on the left and right borders, remember these are 3 bytes wide due to the depth. (One way to do this is check $i\%(xsize * depth) < 3$ and $i\%(xsize * depth) > (xsize * depth - 4)$)

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

iv. Do the convolutions!

v. Note: once the convolutions work, you don't have to copy the resulting sobel_x and sobel_y values back to the CPU (host). You can leave them on the GPU (device), and remove the code from the earlier part of the homework that copied the results there, as they are already on the GPU.

(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. **Optimizing (optional)**

(a) You can try optimizing your code more if you want, but be sure your code is working first. You can do any optimization in the `sobel_cool.cu` file.

(b) You can use `nvprof` to see where your code is spending its time

(c) There's no guarantee anything will help much, but you can try putting things in different memory types (textures? the filters in local or shared memory instead of global?) try using additional compiler flags like `-use_fast_math`. There's a lot more to CUDA than discussed in class.

(d) Try seeing if you can measure some GPU performance events with PAPI.

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