ECE574: Cluster Computing – Homework 3
Convolution

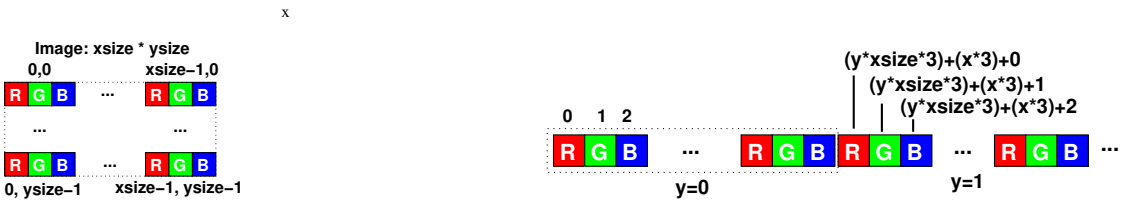
Due: Thursday 9 February 2023, 12:30pm

1. Background

- In this homework we will create some single-threaded convolution code, as found in image processing workloads.
- In a 2D convolution, there is a matrix, often 3x3, that is applied to every pixel in the image. The center of the matrix is applied to the pixel of interest from the input matrix and the surrounding pixels are added together and used to calculate the value for the pixel in the output matrix.

  For example, with a convolution matrix of $\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$ and pixel values $\begin{matrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{matrix}$ if we want to apply the convolution to pixel f, the resulting output pixel would end up being
  $out[1][1] = (0*a)+(-1*b)+(0*c)+(-1*e)+(5*f)+(-1*g)+(0*i)+(-1*j)+(0*k)$

- The provided code uses libjpeg to load an image into memory. It loads a 24-bit color image, meaning each pixel is represented by 3-bytes, one each for red, green and blue as shown in Figure 1a. When doing a convolution, you can treat each color individually. Also these are 8-bit values, so you may need to saturate the result (i.e. make sure the result is not less than 0 or greater than 255). Also recall than in C multi-dimensional arrays are equivalent to one big 1-dimensional array, as shown in Figure 1b.



(a) 2D Pixel layout.

(b) Linear Pixel layout.

Figure 1: Image data memory layout

2. Setup

- Use the same Haswell-EP machine we used for HW#2. If you prefer, you can feel free to develop on your own machine instead, but the code will be tested on Haswell-EP, and also you probably won't have PAPI installed on your own machine.

  As a reminder, use the username handed out in class and ssh in like this

  ```
  ssh -p 2131 username@weaver-lab.eece.maine.edu
  ```

  There's no need to use slurm job submission for this assignment (though you can if you want).

- Download the code template from the webpage. You can do this directly on haswell-ep via
  `wget https://web.eece.maine.edu/~vweaver/classes/ece574/ece574_hw3_code.tar.gz`
  to avoid the hassle of copying it back and forth.

  If you do use scp or sftp to transfer files, remember to set the port to 2131 (and that scp annoyingly uses a CAPITAL P for port, unlike ssh where it is lowercase P).
- Decompress the code
  `tar -xzvf ece574_hw3_code.tar.gz`
- Run `make` to compile the code.

3. **Coding (9 points)**

A template `sobel.c` file is provided. You will only need to modify a few functions as directed.

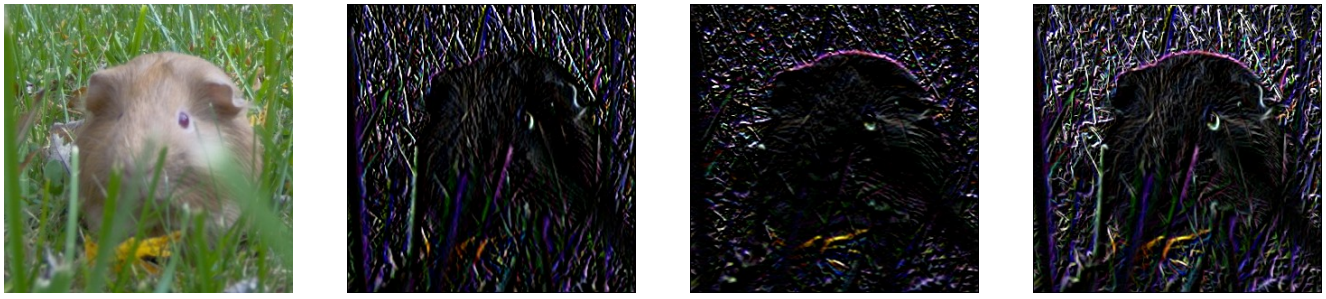
(a) Implement the `generic_convolve()` routine.
This should apply the filter to the pixels in input_image with the result in output_image.
First do this using the sobel_x filter. The matrices to use are provided in the C file.

$$sobel_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

$$sobel_y = \begin{bmatrix} -1 & -2 & 1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Running `./sobel file.jpg` will run on a file named file.jpg with a fixed output of `out.jpg`.
A file `butterfinger.jpg` is provided, with sample results as shown in Figure 2.
Be sure to comment your code!
To get the results to match the provided examples, we use "crop" style output. Start your convolution offset in by 1 pixel, leaving the outside border alone (it will be black).

(b) Once sobel_x is working, then also run sobel_y. Then you will need to combine the x and y results by normalizing it. Also be sure to saturate to fit in 8 bits. Edit the `combine()` routine to implement this. For each pixel in the final output
$out[x][y] = \sqrt{sobel\_x[x][y]^2 + sobel\_y[x][y]^2}$



(a) Input (b) Sobel X (c) Sobel Y (d) Overall Sobel

Figure 2: Example sobel filter results.

(c) Use PAPI to measure the number of last level cache misses and cycles that happen in the two convolves as well as the combine. (`PAPI_L3_TCM` and `PAPI_TOT_CYC`). The provided code already has the `PAPI_library_init()` call for you, but you will have to set up the eventset, add both events, then do start/stop to gather he results. See the Lecture 6 class notes for what needs to be added. All of the PAPI code will be in the `main()` function. So I am looking for 6 values here: `PAPI_L3_TCM` for the first convolve, the second convolve, and the combine. Then the same for `PAPI_TOT_CYC`.

(d) Report in the README file the total time taken for running sobel on the `butterfinger.jpg` file using the `time` command. Also report the cache/cycles PAPI data gathered in the previous step.

(e) Report the same results, but when running the much bigger `space_station_hires.jpg` file.

4. **Debugging**

I know it's a pain developing this when you can't see the resulting jpeg easily. It might be easier if you develop on your own machine before copying it to the haswell machine. Also, if you are running Linux/OSX and have an X11 server available you can ssh in with the `-Y` option and then you can run an image viewer (such as `geeqie`) and it should forward the display to your local display.

If you are having problems with the final output, you can test the interim stages separately, such as modifying the `store_jpeg()` code at the end to just write out `sobel_x` first, until you have that working, and then switch it back to `new_image` when finished.

5. **Something Cool (1 point)**

Copy your working code overtop of `sobel_improved.c`

```
cp sobel.c sobel_improved.c
```

Do one of the following:

- **Optimize the Code**

  Take your code and optimize it in some way.
  This can be something simple like changing loop ordering or unrolling the loops. Alternately you could go for something more complex like converting the code to SIMD code (be aware while this sounds neat it's extremely difficult).

  Use `time` to report the runtime and cache misses as compared to your unoptimized code.

- **Other Convolution Matrix**

  There are many other kinds of convolution Matrix. Look up the matrix values for a non-sobel transformation (such as sharpen, or blur, or edge detect) and modify your code to do that transformation instead. Be sure to note what your code is doing.

- **Other Language**

  Convert your code to be in a non-C language (for example, Fortran or Python). Note in the Fortran case it might be tricky linking against libjpeg.

  Report if the resulting code is faster or slower than the C version.

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 `hw03_submit.tar.gz`. E-mail this file to me.
   - e-mail the file to me by the homework deadline.