ECE574: Cluster Computing – Homework 3
Convolution

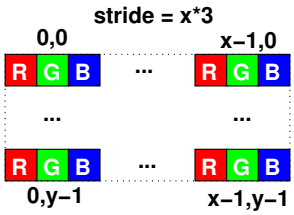
Due: Thursday 1 October 2014, 3:30PM

1. Background

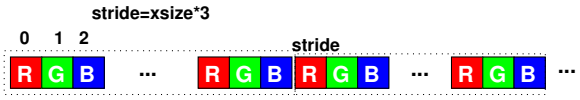
   - In this homework we will create some single-threaded convolution code, as is done in image processing workloads.
   - In a 2D convolution, there is a matrix, often 3x3, that is applied to every pixel in the image. The center of the matrix is applied to the pixel of interest from the input matrix and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

   - For this assignment, log into the same Haswell machine we used for HW2. As a reminder, use the username handed out in class and ssh in like this

     ```
     ssh -p 2131 username@vincent-weaver-2.umelst.maine.edu
     ```

   - Download the code template from the webpage. You can do this directly via

     ```
     wget http://web.eece.maine.edu/~vweaver/classes/ece574_2015f/ece574_hw3_code.tar.gz
     ```

     to avoid the hassle of copying it back and forth.

- Decompress the code
  `tar -xzvf ece574_hw3_code.tar.gz`
- Run `make` to compile the code.

3. **Coding (9 points)**

   (a) Implement the `generic_convolve` routine.
   This should apply the filter to the pixels in image old with the output in image new.
   Initially do this with the sobel_x filter. The matrices to use are in the C file.

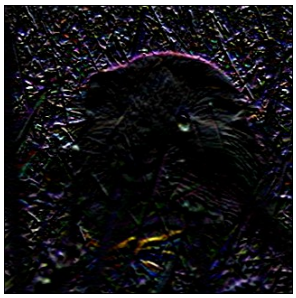
   $$sobel_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

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

   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!

   (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. 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 that happen in your convolve routine (`PAPI_L3_TCM`). See the lecture08 notes for what needs to be added.

   (d) Report the time taken and the cache misses found when running on the provided `IMG_1733.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.

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

   Copy your working code from previously overtop of `sobel_optimized.c`
   `cp sobel.c sobel_optimized.c`

   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 but don't waste too much time on this.

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

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.