CS 33 Fall 2023

Lab 4: Parallel Lab

Due: December 8, 2023 11:59 PM

Pranav Balgi (pgbalgi@cs.ucla.edu) is the lead TA for this lab.

1 Introduction

Welcome to parallel lab! This is the last lab of CS 33 and is quite different from earlier labs. The purpose of this lab is to introduce you to optimization and parallelization on somewhat relevant tasks in the real world.

It is split up into three phases in which you perform different image manipulation tasks. The first phase deals with calculating an average pixel value across all pixels in an image. The second phase deals with converting an image into grayscale as well as recording the maximum grayscale value in the image. The third phase deals with blurring an image using an operation called convolution.

You will not have to come up with algorithms on your own; sequential solutions for these manipulations are provided. Your task is to speed up and parallelize these algorithms. In addition, you are given buggy parallelized code for the first phase which needs to be fixed.

2 Getting Started

The files for this lab are on BruinLearn in the `parallellab-handout.zip` file. You are free to work on this lab on your local machine, but your submission will be graded on the SEASnet server, so be sure to test on there before you submit. If you haven't done so already, please run the following command on the SEASnet server before you test:

```
echo 'export PATH=/usr/local/cs/bin:$PATH' >> ~/.profile && exit
```

The only file you need to modify and submit is `parallel.c`.

3 Evaluation

Your `parallel.c` must compile and produce correct results for each phase. If your solution produces an incorrect result during grading, you will receive 0 points for that phase. To avoid this scenario, make sure your solution does not have any race conditions.

Your solution for each phase will be evaluated based on its speedup with respect to the sequential solution. To assign a score for each phase, we will compare the speedup of your solution to a baseline solution we created. To get maximum points for a phase, your solution's performance must be comparable to the baseline. Approximate baseline speedups for each phase with different image sizes are listed in Table 1. Extra credit will be considered for significant improvement over the baseline on any phase.

Note that speedup can vary greatly between runs, especially when the SEASnet server is under heavy load. Speedup will also vary across different machines, even if they're both SEASnet machines. Therefore, we strongly recommend that you begin testing your solutions early, so you can get a more accurate estimate of your speedups.

Phase	1024x1024 baseline speedup	2048x2048 baseline speedup	4096x4096 baseline speedup	Points
1	3	12	28	20
2	2	7	26	40
3 (3x3 kernel)	6	14	21	40
3 (5x5 kernel)	11	17	23	

Table 1: Approximate baseline speedups of parallel lab phases (tested on `lnxsrv07.seas.ucla.edu`)

4 Testing

To build and test your solution, run the following two commands:

```
make
./test
```

Notice that you must rebuild `test` each time you modify your `parallel.c` file.

You will find that `test` has the following options:

`-h`: Print the command line options

`-p <n>`: Test only phase `n`.

`-r <n>`: Generate image with `n` rows. Use random number of rows if `n` is 0.

`-c <n>`: Generate image with `n` columns. Use random number of columns if `n` is 0.

5 Before Submitting

- Make sure it compiles and passes the tests (without race conditions) on `cs33.seas.ucla.edu`.
- Make sure you have included your name and UID in your `parallel.c` file.

6 Background Information

An image is structured as a 2D array of pixels, with each pixel containing three integer values corresponding to the red, green, and blue channels. RGB pixel values range from 0 to 255.

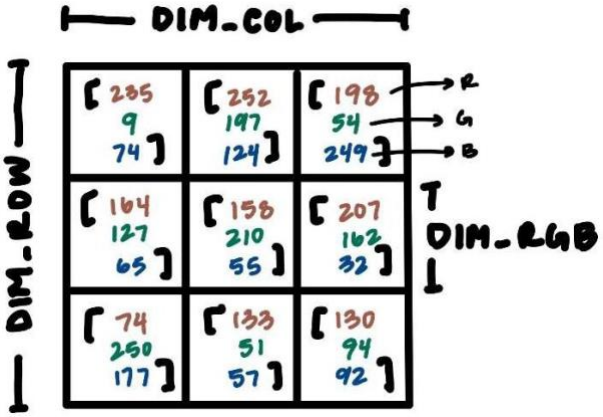

Figure 1: Example RGB image

We randomly generate an image for you in `main.c` and pass it in as a parameter along with the number of rows and columns. The parameters for the input image will look like this:

```
uint8_t img[][NUM_CHANNELS], int num_rows, int num_cols
```

The first dimension of `img` represents the flattened array of pixels and has size `num_rows*num_cols`. The second dimension of `img` represents the RGB channels for each pixel, where `NUM_CHANNELS` is 3.

6.1 Phase 1: Mean Pixel Value

This phase implements a solution for calculating the mean pixel value in an image. Your task is to fix the bug in `mean_pixel_parallel` and optimize its performance. Please refer to `mean_pixel_seq` in `sequential.c` for a correct sequential implementation.

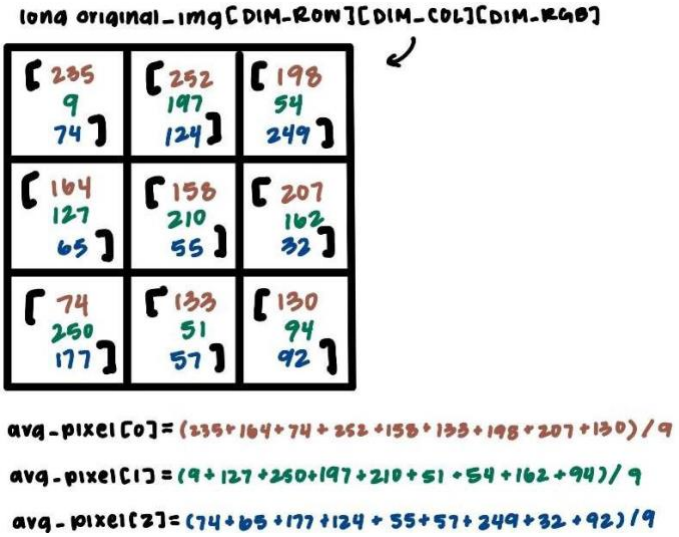

Figure 2: Example of mean pixel value

6.2 Phase 2: Grayscale

This phase implements a solution for converting an image to grayscale. It also computes the maximum grayscale value and the number of times it appears in the grayscale image. The grayscale value of a pixel is

calculated by taking the mean of its RGB values. Each of the RGB values in the grayscale image is set to the grayscale value of the corresponding pixel from the original image.

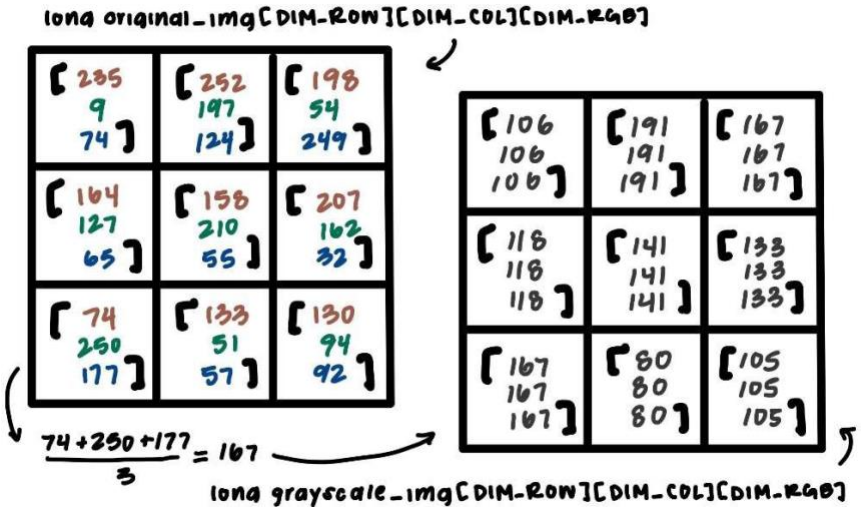

Figure 3: Example of grayscale conversion

6.3 Phase 3: Convolution

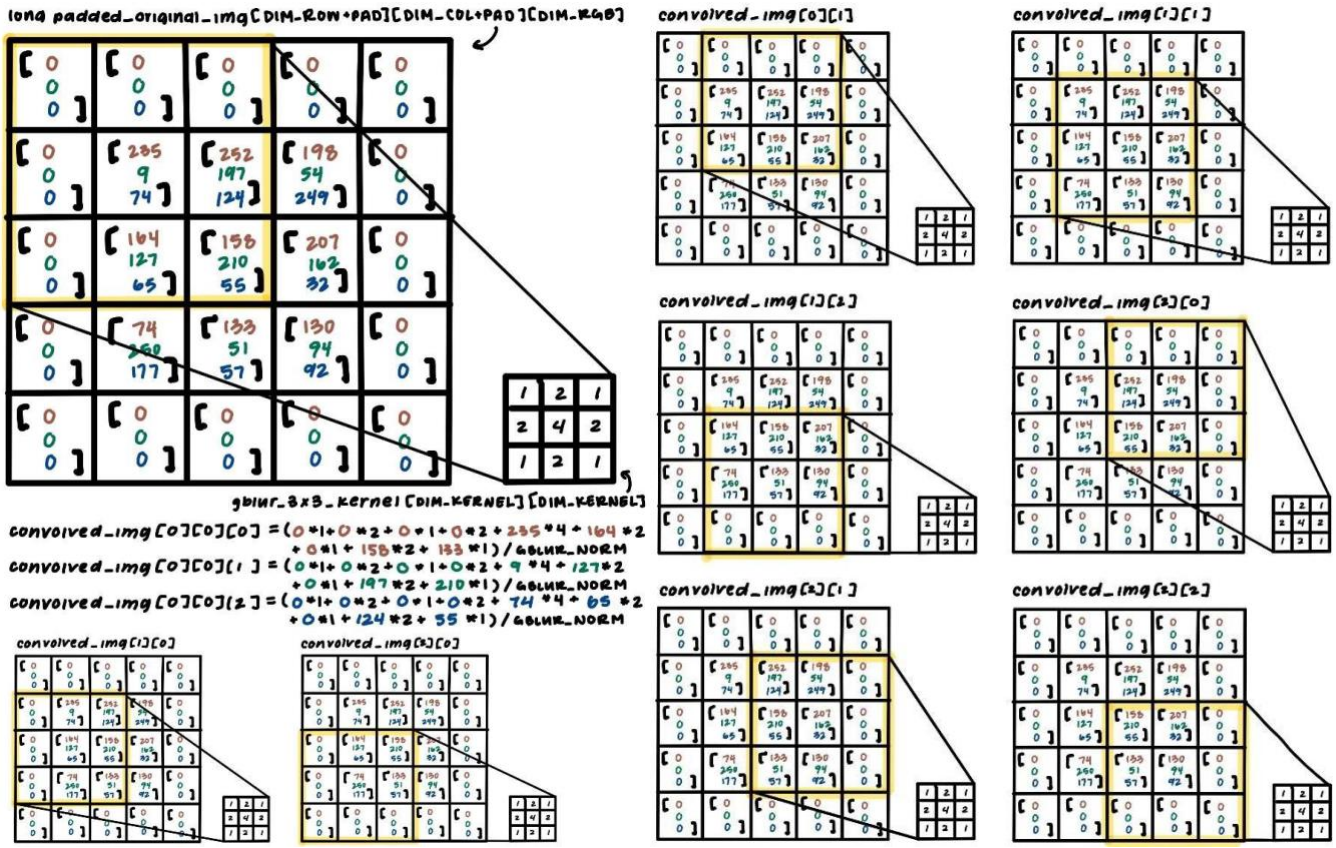

Figure 4: Example of convolution

This phase implements a solution for computing a convolution between a kernel and a padded version of the original image. We don't expect you to know what convolution is, but here are a couple resources if you wish to understand convolution for images:

- https://medium.com/@bdhuma/6-basic-things-to-know-about-convolution-daef5e1bc411
- https://www.youtube.com/watch?v=YgtModJ-4cw

The article shows how adding padding to the original image before convolution will affect the size of the output image. The video does not account for padding but is a better resource to see how a convolution operation is calculated.

For our purposes, we have provided two Gaussian blur kernels (sizes 3x3 and 5x5) for you to test your solutions with. We are assuming the stride for the convolution is 1. Padding is added to ensure that the size of the convolved image is the same as the size of the original image. Note that since the input image is already padded, the convolved image will have fewer rows and columns.

Figure 4 gives a visual example of the calculation for each pixel of the convolved image. Notice how the kernel is applied to each RGB value individually and is stored that way in the convolved image.

7 Final Advice

The optimizations you are expected to implement are based on concepts that have been covered in this class. While the focus of this lab is on OpenMP, it is also worthwhile to think about program optimizations, exploiting instruction level parallelism, and writing cache friendly code.

8 Acknowledgements

This lab was created by Disha Zambani and enhanced by Pranav Balgi.