Assignment 2:

Efficient Program Implementation

In this assignment, you will exercise how to implement a program logic efficiently in C based on system programming concepts.

- Modify the provided code and implement an optimized procedure that performs the same functionality with improved performance.
- Evaluate your program on the Raspberry Pi.
- You should also need to turn in your code that gives the best performance and a short report that describes your optimization strategy.

Due

Submit your implementation before June 14th, Friday, 11:59:59pm, to LMS. We DO NOT allow any late submission.

Description

The provided code reads an image in a BMP format and applies a 3x3 filter over image regions. It's already functional -- it writes a filtered image, e.g., applying blur, sharping, and/or detecting edges.

1. **Background - Convolution filter**
   In image processing, a convolution matrix, also known as kernel, refers to a small matrix. A convolution filter passes over all image pixels to take a dot product of the convolution filter and an image region of the same size, determining a pixel of the output image.
   More formally, an image can be represented as a set of pixels in a 2D space, saying $f(x, y)$ where x and y are coordinates. A general expression of a convolution is

   $$g(x, y) = \omega * f(x, y) = \sum_{dx=-a}^{a} \sum_{dx=-b}^{b} \omega(dx, dy) f(x + dx, y + dy)$$

   where $g(x, y)$is the output filtered image, ωis the convolution matrix, and every element in the matrix is considered by $-a \leq dx < a$ and $-b \leq dx < b$.

   In a nutshell, you can imagine the convolution process as follows:

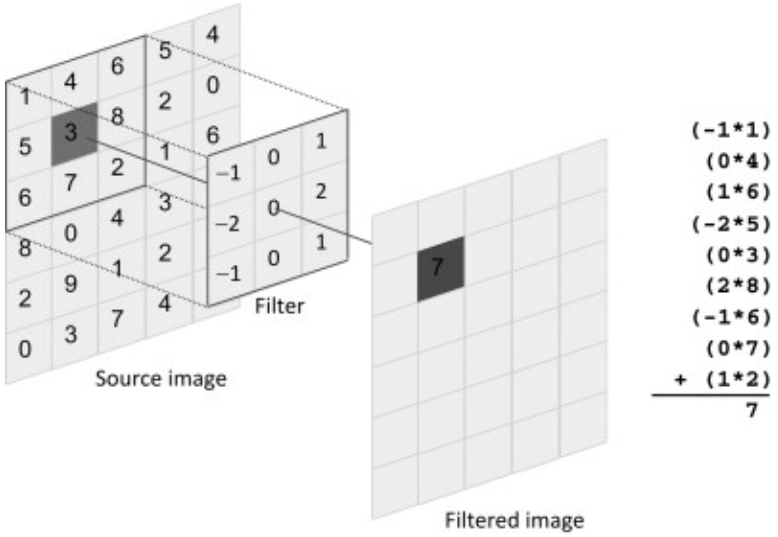

   Image credit: https://pranav-ap.netlify.app/posts/cnn/
   (This convolution procedure applies for all 3x3 regions of the input image to compute all pixels of the output image.)

   The convolution usually needs to handle values at edges -- pixels outside of the image boundaries. Many various methods exist; we will use a very simple way -- assume that pixels in the boundary have all zero values.

   With different kernels, you can create different filtered images. For example, imagine a 3x3 filter which fills with 1/9. This convolution kernel averages pixels in a given position, creating a blurred image. Please see different kernels and created images in Wikipedia: https://en.wikipedia.org/wiki/Kernel_(image_processing)

2. Background - BMP format

The BMP file format, also known as the bitmap image file, is an image file format especially used on Microsoft Windows and OS/2 operating systems. It starts with headers, which contain various metadata information for the image. It also contains a pixel array region, which has 2D pixels in an (usually) uncompressed form, where each pixel has red, color, and blue values, 1 byte for each, for 24-bit images.

3. Provided program

The provided program opens and parses a BMP image to extract the two-dimensional pixel array region. It then applies a 3x3 filter, defined in main.c, through convolution, creating an output image. Since we consider a 3-channel RGB pixel image, it applies the kernels for each color value.

In this assignment, you only need to touch the convolution computation process, which is the most computationally expensive part of this program. The baseline implementation is already given with "filter_baseline()" which calls "convolution()". This implementation looks okay and functional -- you can even test different convolution filters, creating different filtered images. However, note that it does not fully utilize the concept of *good system programming*.

4. What to do

Your job is to optimize this baseline procedure and create your own implementation, "filter_optimized()" in "hw2.c", which performs the same functionality with better performance. You don't need to rely on anything provided in "hw2.c", meaning that you are allowed to change anything as long as it produces the same output as "filter_baseline()."

The given source code runs two convolution implementations, one for the baseline (filter_baseline) and one for your implementation (filter_optimized), to measure the speedup comparing the two. We will use the execution time for the comparison metric, and the measurement code is already ready.

Please consider what you have learned in the course -- memory access patterns, caches, the rough number of instructions executed, etc. There is no single answer. Submit the best implementation you make -- the fastest implementation in terms of the execution time.

5. What to note

- You are going to use the gcc compiler with the `-O0` option (no compile-time optimization.)
- Your implementation should run with any 3x3 convolution matrix.
- You can assume that the image width is a multiple of 32 pixels. (The provided BMP loader only handles these cases.)
- The measurement may run each of filter_baseline() and filter_optimized() multiple times. It avoids noises during the measurement and makes the time measurement stable.
    - The speedup would not be perfectly measured if the image file is too small since the execution time could be too short.
    - So, to obtain a bit more representative speedup results of your implementation, please use a large image file, e.g., img_1024.bmp provided with the source code.
- We will assume that your implementation runs on a Linux environment using gcc.
- Do not modify provided files, other than hw2.c.
- Don't use multi-threading for this assignment.
- Don't use SIMD (ARM NEON) instructions for fair comparison.

Submission

Submit a zip file that has your **source code** and **report**. Before the submission, name the zip file with this format: hw2_STUDENTID.zip (e.g., hw2_200012345.zip)

Source code

- You have to submit a single file, hw2.c. No makefile changes are allowed. If you believe you should modify it, please contact the instructors.
- You don't need to upload other files. Your implementation should run with the other original files provided.
- You can include other library headers if needed.
- The name of your source code file has to be "hw2.c".

Report

- Your report should be one or two pages. (No 3-page or more). Include your name and student ID on the first page. Use an 11-pt Times font, and write in English (other than your name.)
- The section should be organized as follows:
- **First section** - Implementation results: You should report the speedup of your best implementation for various image sizes as compared to the provided baseline. Use a table or a graph to show them.
- **Second section** - Optimization approach: **You should discuss various aspects of your optimization approaches.** The typical examples are:
    - Describe your optimization strategy and discuss it with evaluated results. If you have multiple strategies and combine them to make the best one, discuss the speedup breakdown, i.e., how much speedup is achieved by each strategy.
    - If you tried some strategies but failed to improve your performance, please also discuss them.
    - If needed, include figures and/or graphs to help your explanation.
    - Don't put your code into the report. Please try to explain your high-level ideas and approaches without explaining the code line by line.
- The name of your report file has to be "report.pdf."