Project: Program Optimization

In this project, you will exercise how to optimize a C program based on system programming concepts.

- Modify the provided code and implement an optimized procedure, which performs the same functionality with the improved performance.
- You should also need to turn in the report that describes your optimization strategy.

Due

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

Project 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 is a coordinate. 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 $-a \le dx < a$ and $-b \le dx < b$.

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

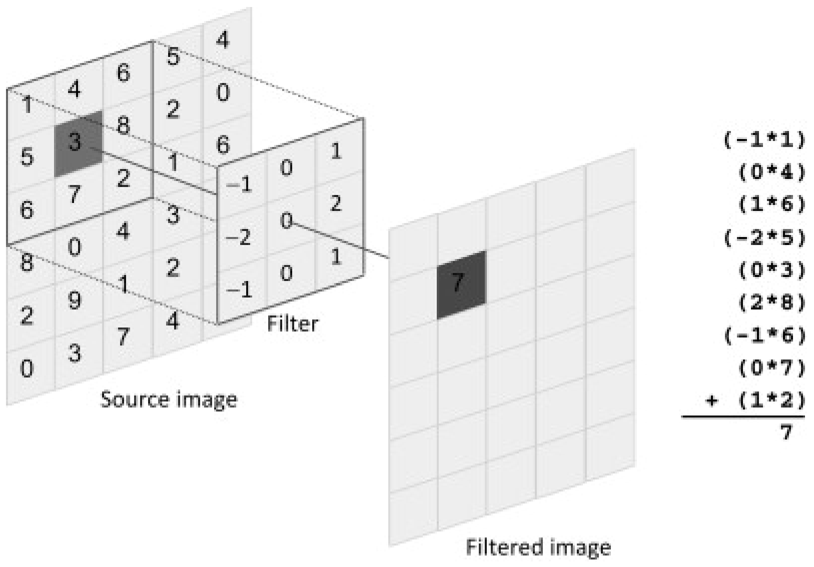

   Image credit: https://pranav-ap.netlify.app/posts/cnn/
   (This convolution procedure applies for all 3x3 regions of the input image to compute all pixels of the output image.)

   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 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 project, 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’s not fully utilizing 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 “proj.c”, which performs the same functionality with better performance. You don’t need to rely on anything provided in “proj.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. For the comparison metric, we will use Cycles Per Element (CPE), and the measurement code is also already ready.

   Please consider what you have learned in the course -- memory access pattern, caches, code motion, etc. There is no single answer. Submit the best implementation you make -- the fastest implementation in terms of CPE.

   In particular, this program is greatly improved by using SIMD. If you decide to use SIMD, you need to submit two files, one optimized implementation without SIMD and the other one using SIMD. Please read the submission section below.

5. **What to note**

- You are going to use the gcc compiler with -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 cycle measurement stable.
- Do not modify provided files, including Makefile, other than proj.c.
- We will assume that your implementation runs on a Linux environment using gcc.

Grading Environment

We will use a grading environment to test all the submitted implementations. To give chances to run your code on the grading machine in advance, we will provide a submission script soon. Stay tuned!

- Note that we will refer to the speedup shown in your report (i.e., running on your own machine) as well as the speedup on the grading machine.

Submission

Submit a zip file that has your **source code** and **report**. Before the submission, name the zip file with this format: proj_STUDENTID.zip (e.g., proj_200012345.zip)

Source code
- You have to submit a single file, proj.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, including header files for SIMD intrinsics.
- The name of your source code file has to be "proj.c".
- If your optimization strategy utilizes SIMD, please submit two files, one including optimized implementation with SIMD, and the other one without using SIMD. Name your source code, "proj.c" and "proj_wo_SIMD.c", respectively.

Report
- Your report should be up to 3 pages. (No 4-page report). Include your name and student ID in the first page. Use an 11-pt Times font, and write in English (other than your name.)
- In the first page, you should also mention the speedup of your best implementation for various image sizes, as compared to the provided baseline.
- 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 the performance, please also discuss them.
- If needed, include figures and/or graphs to help your explanation.
- The name of your report file has to be “report.pdf.”