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 12th, Sunday, 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 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:

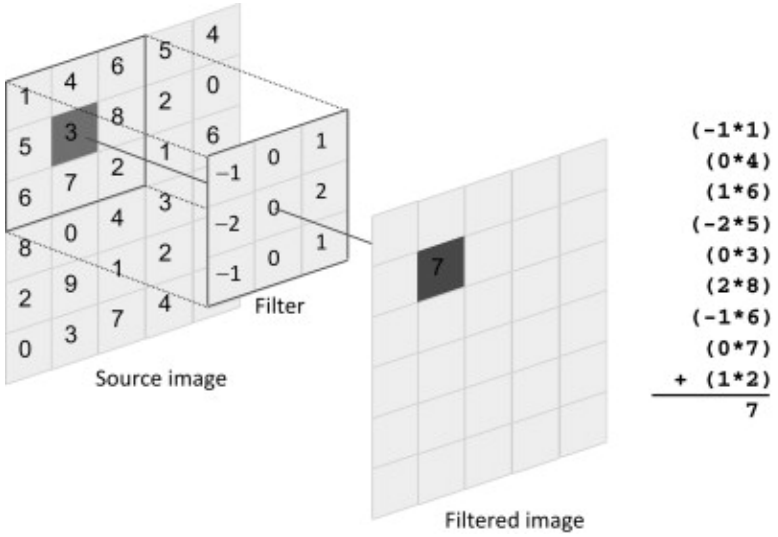

   Image credit: https://pranav-ap.netlify.app/posts/cnn/
   (This convolution procedure applies for all 3x3 regions of the input image to compute all pixels of the output image.)

   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 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. 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, code motion, etc. There is no single answer. Submit the best implementation you make -- the fastest implementation in terms of the execution time.

In particular, this program is greatly improved by using SIMD (The processor vendors use their own names for the corresponding technologies depending on the architecture, like Intel SSE/AVX, ARM NEON.) Although we cannot fully cover SIMD during the lecture, you can learn and try to use it. 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 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, including Makefile if possible, other than proj.c.
    - If you are an ARM user, you may need to change the Makefile depending on the settings. Please see below.

For ARM Users

The provided source code is not fully tested across all available ARM machines. So please be aware of the following things.

- For C files:
  I believe there would be no issues to run the source code as long as it is compiled -- it does not have any architecture-dependent implementations. However, if you find any issue in the provided C file when running on your machine, please email me.
- For Makefile:
  The provided Makefile is designed to run on Intel architectures. So, you would be in trouble during the compilation due to your microarchitecture. In such cases, change the following line of the Makefile to comment out the architecture-dependent option (`-mfpmath=sse`):

```
CFLAGS = -Wall -O0 -march=native -fno-tree-vectorize #-mfpmath=sse
```

    - If the compilation issue persists even with this fix, please email me.
    - You may already notice that this fix may disable the option required to use SIMD libraries.
        - If the fix makes the compiler run correctly and your implementation does not have any SIMD-related implementation, you're good to go.
        - If you also implement a version that uses SIMD, you should submit the file separately in LMS as aforementioned. Please also submit your Makefile along with the SIMD-version source code file.

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 provide a submission script written in Python: client.py. (I tested the submission script with Python 3.8.5; but believe that it would also work most of the up-to-date Python versions.)
You can use the python script like this:

```
~/…../proj_server$ python client.py proj.c
Connecting to the server
Sending the file
Transfer completed: 1820 Bytes
Waiting for the result. It may take a while.
Currently, 0 task(s) left
rm -f bmplib.o perfenv.o main.o proj.o bmpfilter
gcc -Wall -O0 -march=native -fno-tree-vectorize -mfpmath=sse   -c -o bmplib.o
bmplib.c
gcc -Wall -O0 -march=native -fno-tree-vectorize -mfpmath=sse   -c -o perfenv.o
perfenv.c
gcc -Wall -O0 -march=native -fno-tree-vectorize -mfpmath=sse   -c -o main.o
main.c
gcc -Wall -O0 -march=native -fno-tree-vectorize -mfpmath=sse   -c -o proj.o
proj.c
gcc -Wall -O0 -march=native -fno-tree-vectorize -mfpmath=sse bmplib.o perfenv.o
main.o proj.o -lm -o bmpfilter
BMP file loaded: 1024 X 1024
Trial 0
Trial 1
Trial 2
Trial 0
Trial 1
Trial 2
Your speedup:   0.998705
BMP file loaded: 128 X 128
Trial 0
Trial 1
Trial 2
Trial 0
Trial 1
Trial 2
Your speedup:   0.999566
```

- This script will (i) upload the specified file, e.g., proj.c, (ii) compile with the same source code provided, (iii) run it, and then (iv) show the printed results of speedup if running on the Grading environment.
- The script will wait automatically if there're multiple pending jobs on the server.
- Note that we will also refer to the speedup shown in your report (i.e., running on your own machine) as well as the speedup on the grading machine that we will measure on the grading machine.
- If you believe that the server shuts down for any reason, please email me.

Processor configurations of the grading machine

- The main memory size is 128GB. / The gcc version is 9.4.0.

```
vendor_id       : GenuineIntel
cpu family      : 6
model           : 85
model name      : Intel(R) Xeon(R) Silver 4110 CPU @ 2.10GHz
stepping        : 4
microcode       : 0x2006b06
cpu MHz         : 2100.000
cache size      : 11264 KB
physical id     : 1
siblings        : 16
core id         : 7
cpu cores       : 8
apicid          : 31
initial apicid  : 31
fpu             : yes
fpu_exception   : yes
cpuid level     : 22
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36
clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp lm constant_tsc
art arch_perfmon pebs bts rep_good nopl xtopology nonstop_tsc cpuid aperfmperf pni
pclmulqdq dtes64 ds_cpl vmx smx est tm2 ssse3 sdbg fma cx16 xtpr pdcm pcid dca sse4_1
sse4_2 x2apic movbe popcnt tsc_deadline_timer aes xsave avx f16c rdrand lahf_lm abm
3dnowprefetch cpuid_fault epb cat_l3 cdp_l3 invpcid_single intel_ppin ssbd mba ibrs
ibpb stibp tpr_shadow vnmi flexpriority ept vpid ept_ad fsgsbase tsc_adjust bmi1 hle avx2
smep bmi2 erms invpcid rtm cqm mpx rdt_a avx512f avx512dq rdseed adx smap
clflushopt clwb intel_pt avx512cd avx512bw avx512vl xsaveopt xsavec xgetbv1 xsaves
cqm_llc cqm_occup_llc cqm_mbm_total cqm_mbm_local dtherm ida arat pln pts hwp
hwp_act_window hwp_epp hwp_pkg_req pku ospke md_clear flush_l1d
vmx flags       : vnmi preemption_timer posted_intr invvpid ept_x_only ept_ad ept_1gb
flexpriority apicv tsc_offset vtpr mtf vapic ept vpid unrestricted_guest vapic_reg vid ple
shadow_vmcs pml ept_mode_based_exec tsc_scaling
bugs            : cpu_meltdown spectre_v1 spectre_v2 spec_store_bypass l1tf mds swapgs taa
itlb_multihit
bogomips        : 4201.69
clflush size    : 64
cache_alignment : 64
address sizes   : 46 bits physical, 48 bits virtual
```

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.
    - In this case, please also specify the (micro)architecture in your report used during the implementation. We will use the system suitable for your architecture-dependent implementation for grading.

Report

- Your report should be up to 3 pages. (No 4-page report or more). 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.
- **The main purpose of this report is to 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.
        - That may include some trials to use your various optimization approaches or SIMD, etc.
    - If needed, include figures and/or graphs to help your explanation.
- The name of your report file has to be "report.pdf."