Performance Lab

1 Introduction

This assignment focuses on optimizing memory-intensive code. Image processing provides numerous examples of functions that can benefit from optimization. In this lab, we will explore two image processing operations: **pinwheel**, which rotates an image counter-clockwise by 90 degrees, and **motion**, which "blurs" an image.

In this lab, we will represent an image as a two-dimensional matrix M, where M[i][j] denotes the value of the (i, j)th pixel of M. Pixel values are triples of red, green, and blue (RGB) values. We will only consider square images. Let N represent the number of rows (or columns) of an image. Rows and columns are numbered in C-style, ranging from 0 to N - 1.

Given this representation, the pinwheel operation can be easily implemented as a combination of the following two matrix operations:

- Transpose: For each (i, j) pair, M[i][j] and M[j][i] are interchanged.
- Exchange rows: Row i is exchanged with row N - 1 - i.

This combination is illustrated in Figure 1.

The motion operation is implemented by replacing each pixel value with the average of all the surrounding pixels (within a maximum 3x3 window centered at that pixel). As illustrated in Figure 2, the values of pixels M2[1][1]and M2[N-1][N-1] are provided below:

$$M2[1][1] = (\sum_{i=0}^{2}\sum_{j=0}^{2} M1[i][j]) \div 9 \tag{1}$$

$$M2[N-1][N-1] = (\sum_{i=N-2}^{N-1}\sum_{j=N-2}^{N-1} M1[i][j]) \div 4 \tag{2}$$

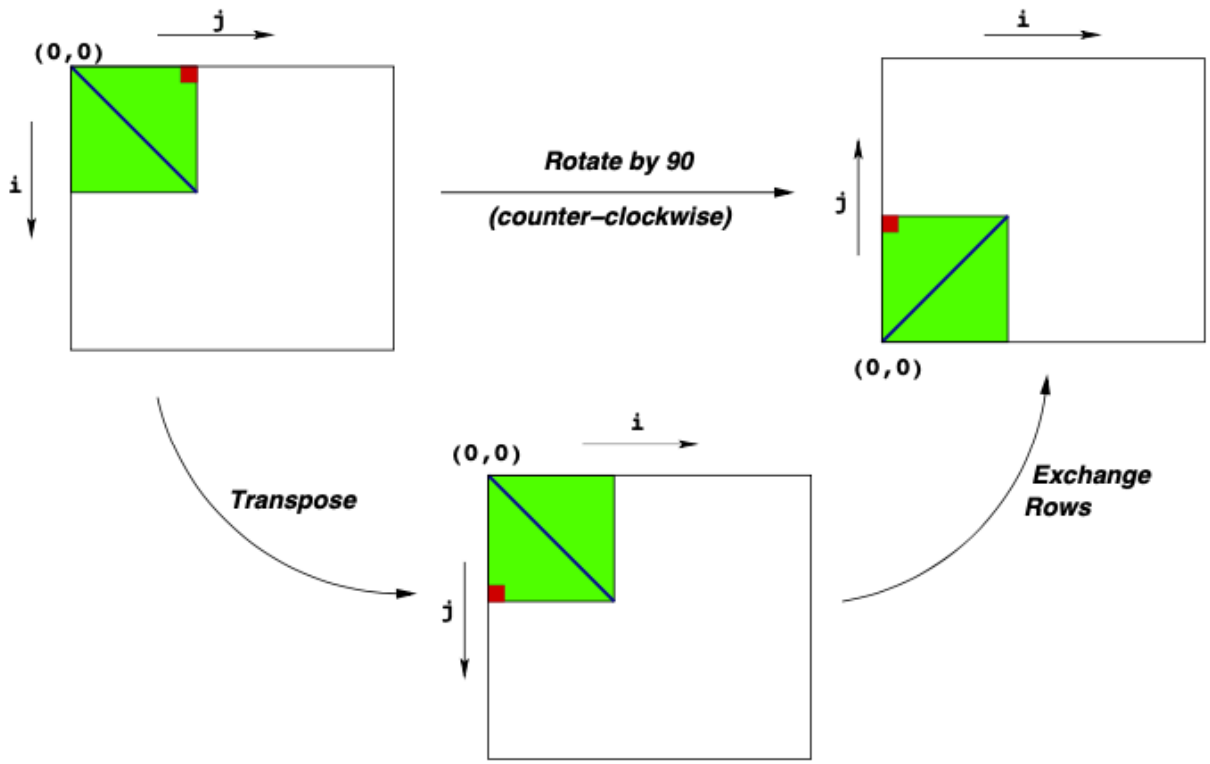

Figure 1: Rotation of an image by 90◦ counterclockwise

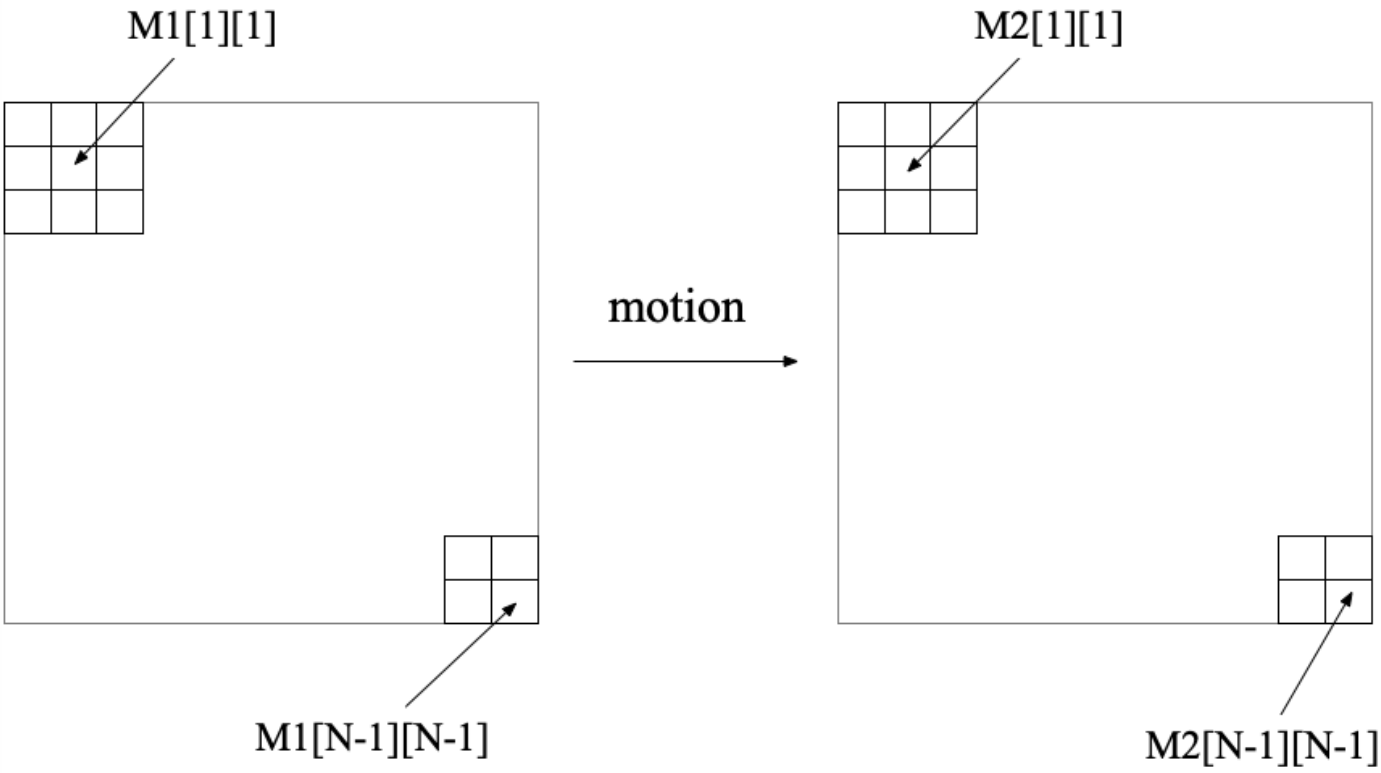

Figure 2: blur an image

2 Hand Out Instructions

Start by copying perflab-handout-student.zip to a protected directory in which you plan to do your work. Then give the command: unzip perflab-handout-student.zip. This will cause a number of files to be unpacked into the directory. The only file you will be modifying and handing in is kernels.c. The driver.c program is a driver program that allows you to evaluate the performance of your solutions. Use the command make driver to generate the driver code and run it with the command ./driver.
Looking at the file kernels.c you'll notice a C structure team into which you should insert the requested **identifying information** about the one or two individuals comprising your programming team. **Do this right away so you don't forget.**

Data Structures

The core data structure deals with image representation. A pixel is a struct as shown below:

```
typedef struct {
 unsigned short red; /* R value */ *

*unsigned short green; /* G value */ *

*unsigned short blue; /* B value */

} pixel;
```

As can be seen, RGB values have 16-bit representations ("16-bit color"). An image I is represented as a one-dimensional array of pixels, where the (i, j)th pixel is I[RIDX(i,j,n)]. Here n is the dimension of the image matrix, and RIDX is a macro defined as follows:

```
\#define RIDX(i,j,n) ((i)*(n)+(j)) See the file defs.h for this code.
```

pinwheel
The following C function computes the result of rotating the source image src by 90◦ and stores the result in destination image dst. dim is the dimension of the image.

```
void naive_pinwheel(int dim, pixel *src, pixel *dst) {
  int i, j;
  for(i=0; i < dim; i++)
    for(j=0; j < dim; j++)
      dst[RIDX(dim-1-j,i,dim)] = src[RIDX(i,j,dim)];
  return;
 }
```

The above code scans the rows of the source image matrix, copying to the columns of the destination image matrix. Your task is to rewrite this code to make it run as fast as possible using techniques like code motion, loop unrolling and blocking.
See the file kernels.c for this code.

motion

The motion function takes as input a source image src and returns the blurred result in the destination image dst. Here is part of an implementation:

```
void naive_motion(int dim, pixel *src, pixel *dst) {
  int i, j;
  for(i=0; i < dim; i++)
    for(j=0; j < dim; j++)
      dst[RIDX(i,j,dim)] = avg(dim, i, j, src); /* Blur the (i,j)th pixel */
  return;
}
```

The function avg returns the average of all the pixels around the (i,j)th pixel. Your task is to optimize smooth (and avg) to run as fast as possible. (*Note:* The function avg is a local function and you can get rid of it altogether to implement motion in some other way.)

This code (and an implementation of avg) is in the file kernels.c.

3 Performance measures

```
$ ./driver
Student: 11111111
Email: 11111@seu.edu.cn

pinwheel: Version = naive_pinwheel: Naive baseline implementation:
Dim             64      128     256     512     1024    Mean
Your CPEs       2.0     2.5     4.8     9.7     8.5
Baseline CPEs   2.1     2.6     4.7     8.7     8.9
Speedup         1.1     1.1     1.0     0.9     1.0     1.0

pinwheel: Version = optimized_pinwheel: Optimized implementation using blocking:
Dim             64      128     256     512     1024    Mean
Your CPEs       1.9     2.0     2.2     3.2     3.7
Baseline CPEs   2.1     2.6     4.7     8.7     8.9
Speedup         1.1     1.3     2.1     2.8     2.4     1.8

pinwheel: Version = pinwheel: Current working version:
Dim             64      128     256     512     1024    Mean
Your CPEs       2.4     2.5     2.2     2.5     3.7
Baseline CPEs   2.1     2.6     4.7     8.7     8.9
Speedup         0.9     1.0     2.1     3.4     2.4     1.7

motion: Version = motion: Current working version:
Dim             32      64      128     256     512     Mean
Your CPEs       11.2    12.2    12.5    12.5    13.1
Baseline CPEs   43.0    43.0    43.1    44.4    44.5
Speedup         3.8     3.5     3.5     3.5     3.4     3.5

motion: Version = naive_motion: Naive baseline implementation:
Dim             32      64      128     256     512     Mean
Your CPEs       44.2    44.2    44.3    43.2    44.8
Baseline CPEs   43.0    43.0    43.1    44.4    44.5
Speedup         1.0     1.0     1.0     1.0     1.0     1.0

motion: Version = optimized_motion: Optimized implementation:
Dim             32      64      128     256     512     Mean
Your CPEs       11.1    12.3    12.3    13.5    13.7
Baseline CPEs   43.0    43.0    43.1    44.4    44.5
Speedup         3.9     3.5     3.5     3.3     3.2     3.5

Summary of Your Best Scores:
  Pinwheel: 1.8 (optimized_pinwheel: Optimized implementation using blocking)
  Motion: 3.5 (motion: Current working version)
```

4.Infrastructure

We have provided support code to help you test the correctness of your implementations and measure their performance. This section describes how to use this infrastructure. The exact details of each part of the assignment is described in the following section.

Note: The only source file you will be modifying is kernels.c.

Versioning

You will be writing many versions of the pinwheel and motion routines. To help you compare the performance of all the different versions you've written, we provide a way of "registering" functions.

For example, the file kernels.c that we have provided you contains the following function:

```
void register_pinwheel_functions() {
  add_pinwheel_function(&pinwheel, pinwheel_descr);
}
```

This function contains one or more calls to add pinwheel function. In the above example, add pinwheel function registers the function pinwheel along with a string pinwheel descr which is an ASCII description of what the function does. See the file kernels.c to see how to create the string descriptions. This string can be at most 256 characters long.

A similar function for your motion kernels is provided in the file kernels.c.

Driver

The source code you will write will be linked with object code that we supply into a driver binary. To create this binary, you will need to execute the command

```
unix> make driver
```

You will need to re-make driver each time you change the code in kernels.c. To test your implementations, you can then run the command:

```
unix> ./driver
```

The driver can be run in four different modes:

- *Default mode*, in which all versions of your implementation are run.
- *Autograder mode*, in which only the pinwheel() and motion() functions are run. This is the mode we will run in when we use the driver to grade your handin.
- *File mode*, in which only versions that are mentioned in an input file are run.
- *Dump mode*, in which a one-line description of each version is dumped to a text file. You can then edit this text file to keep only those versions that you'd like to test using the *file mode*. You can specify whether to quit after dumping the file or if your implementations are to be run.

  If run without any arguments, driver will run all of your versions (*default mode*). Other modes and options can be specified by command-line arguments to driver, as listed below:

  -g : Run only pinwheel() and motion() functions (*autograder mode*).

  -f : Execute only those versions specified in (*file mode*).

  -d : Dump the names of all versions to a dump file called , *one line* to a version (*dump mode*).

  -q : Quit after dumping version names to a dump file. To be used in tandem with -d. For example, to quit immediately after printing the dump file, type ./driver -qd dumpfile.

  -h : Print the command line usage.

5.Assignment Details

Optimizing pinwheel (50 points)

Optimizing motion (50 points)

In this part, you will optimize pinwheel and motion to achieve as low a CPE as possible.
For example, running driver with the supplied naive version generates the output shown below:

```
pinwheel: Version = pinwheel: Current working version:
Dim           64  128 256 512 1024  Mean
Your CPEs     2.4 2.5 2.2 2.5 3.7
Baseline CPEs 2.1 2.6 4.7 8.7 8.9
Speedup       0.9 1.0 2.1 3.4 2.4   1.7
motion: Version = motion: Current working version:
Dim           32    64    128   256   512   Mean
Your CPEs     11.2  12.2  12.5  12.5  13.1
Baseline CPEs 43.0  43.0  43.1  44.4  44.5
Speedup       3.8   3.5   3.5   3.5   3.4   3.5
```

Coding Rules

You may write any code you want, as long as it satisfies the following:
• It must be in ANSI C. You may not use any embedded assembly language statements.

• It must not interfere with the time measurement mechanism. You will also be penalized if your code prints any extraneous information.

You can only modify code in kernels.c. You are allowed to define macros, additional global variables, and other procedures in these files.

Evaluation

Your solutions for pinwheel and motion will each count for 50% of your grade. The score for each will be based on the following:

• Correctness: You will get NO CREDIT for buggy code that causes the driver to complain! This includes code that correctly operates on the test sizes, but incorrectly on image matrices of other sizes. As mentioned earlier, you may assume that the image dimension is a multiple of 32.

• CPE:You will receive full credit for your implementations of pinwheel and motion if they are correct and achieve mean CPEs above thresholds of 1.5 and 3.0, respectively. You will get 60% of the full credit if you achieve mean CPEs above thresholds of 1.2 and 2.5, respectively. Additionally, the top five students will receive extra points ranging from 1 to 5.