CS 135, Fall 2010
Project 4: Code Optimization
Assigned: November 30th, 2010
Due: December 12,, 2010, 12noon

1 Introduction

This assignment deals with optimizing memory intensive code. Image processing offers many examples of functions that can benefit from optimization. In this project, we will consider two image processing operations: `rotate`, which rotates an image counter-clockwise by $90°$, and `smooth`, which "smooths" or "blurs" an image. You have probably used these operations when dealing with images (in Photoshop, Paint, etc.).

For this project, we will consider an image to be represented as a two-dimensional matrix M, where $M_{i,j}$ denotes the value of (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 denote the number of rows (or columns) of an image. Rows and columns are numbered, in C-style, from 0 to $N-1$.

Given this representation, the `rotate` operation can be implemented quite simply as the 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 `smooth` operation is implemented by replacing every pixel value with the average of all the pixels around it (in a maximum of 3×3 window centered at that pixel). Consider Figure 2. The values of pixels `M2[1][1]` and `M2[N-1][N-1]` are given below:

$$M2[1][1] = \frac{\sum_{i=0}^{2} \sum_{j=0}^{2} M1[i][j]}{9}$$

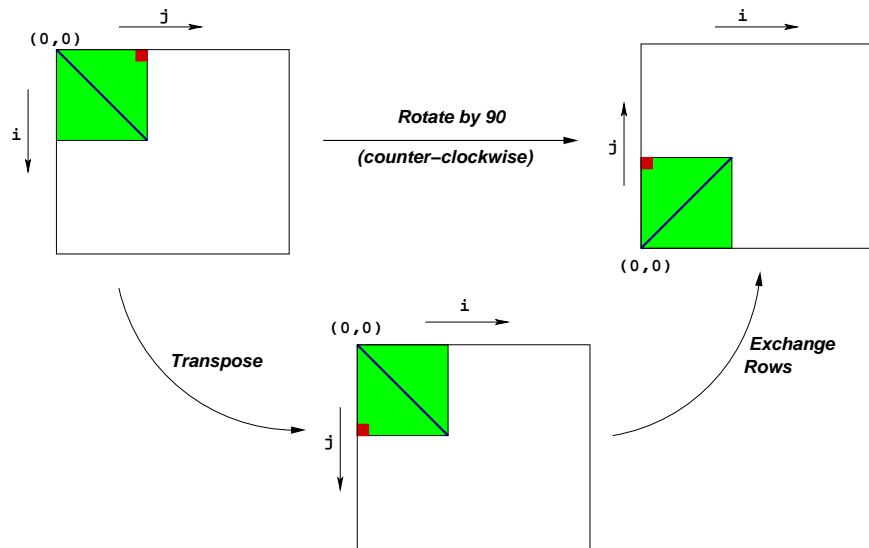$$M2[N-1][N-1] = \frac{\sum_{i=N-2}^{N-1} \sum_{j=N-2}^{N-1} M1[i][j]}{4}$$
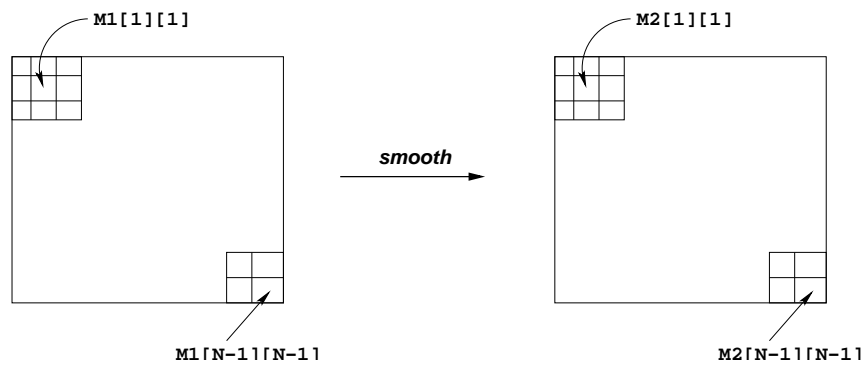
Figure 1: Rotation of an image by $90°$ counterclockwise

Figure 2: Smoothing an image

2   Logistics

You **must** work alone in solving the problems for this project. The only "hand-in" will be electronic. Any clarifications and revisions to the assignment will be posted on the course Web page. There is NO collaboration of any sort allowed on this project – this includes outside sources (online or otherwise) and you cannot discuss solution strategies with other students. However, you can use any/all of the notes (and solutions) from the in-class discussion of team assignment 8 – it is directly relevant to project 4.

The project source files and installation will be discussed in the lab sessions during the week of November 30th. Some of the project logistics will be discussed in the lectures during the week of Dec. 1st. Any questions regarding this project will be addressed **only** during the labs (and lectures) of the week of Nov.30th and Dec.2nd.

3   Hand Out Instructions

Links, to the code, will be provided on the homeworks web page and more details can be obtained in the labs. Start by copying `project4.tar` to a protected directory in which you plan to do your work. Then give the command: `tar xvf project4.tar`. This will cause a number of files to be unpacked into a directory called project4. 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 yourself. **Do this right away so you don't forget.** (This is similar to project 1.)

4   Implementation Overview

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 `pixel`s, 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.

Rotate

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_rotate(int dim, pixel *src, pixel *dst) {
  int i, j;
   for(j=0; j < dim; j++)
     for(i=0; i < dim; i++)
      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 some of the techniques we discussed in the lecture.

See the file `kernels.c` for this code.

Smooth

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

```
void naive_smooth(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); /* Smooth 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 `smooth` in some other way.)

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

Performance measures

Our main performance measure is the number of (processor) cycles it takes to run for an image of size $N \times N$ for different values of N.

The ratios (speedups) of the execution time of the optimized implementation over the naive one will constitute a *score* of your implementation.

Assumptions

To make life easier, you can assume that N is a multiple of 32. Your code must run correctly for all such values of N, but we will measure its performance only for large sizes (of 512 or greater).

5 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. The code handed out will test and measure the performance of your solutions. We used the system `clock` utility to measure the time – as a result, you can get different times each time you run the code. The performance measurement also takes the average of running your code 10 times. So you do not need to test multiple times (although you are not discouraged from doing so).

Note: The only source file you will be modifying is kernels.c; in fact you will only be writing the code for my_rotate and my_smooth.

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
```

Note: 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
```

System Details

Your solutions will be evaluated on the SEAS Hobbes server. You MUST use these machines to measure the performance of your solutions. You MUST use the gcc compiler, with no compiler optimization options. It is important to note that we will only be using the hobbes system to measure the performance of your solutions – improvements made on any other system will not be taken into account during our grading.

Team Information

Important: Before you start, you should fill in the struct in `kernels.c` with information about your name etc. This information is just like the one for the first project.

6 Assignment Details

Optimizing Rotate

In this part, you will optimize `rotate` to achieve as low a time (cycles) as possible. You should compile `driver` and then run it with the appropriate arguments to test your implementations. The target performance improvement (*i..e,* how much faster your version runs when compared with the naive version handed out) is at least 15% for image sizes 1024 and higher on the SEAS Hobbes server. This only constitutes a minimum (i.e., a grade of 70% on the project), so you must try to get an improvement beyond this number by trying out different optimizations that could improve performance.

Optimizing Smooth

In this part, you will optimize `smooth` to achieve as low a time (cycles) as possible. The target performance improvement (*i..e,* how much faster your version runs when compared with the naive version handed out) is 15% for image sizes 1024 and higher on the SEAS Hobbes server. This only constitutes a minimum (i.e., a grade of 70% on the project), so you must try to get an improvement beyond this number.

Hints: (Some tips will be discussed in the class lectures.) Focus on optimizing the inner loop (the code that gets repeatedly executed in a loop). The `smooth` is more compute-intensive and less memory-sensitive than the `rotate` function, so the optimizations are of somewhat different flavors.

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 MUST use ONLY the optimizations we discussed in the lectures/course. *If you use any other optimizations then you will receive a zero on this project.*

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

Evaluation

This project is worth 50 points towards your overall project grade. You will also be able able to earn more than 50 points if you use more optimizations and produce significant speedups over the naive version. If you choose to optimize only one of the two functions (Rotate and Smooth), then you can earn a maximum of upto 30 points depending on how well you have optimized the code.

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.

- **Performance:** in terms of the number of processor cycles taken by your code. The minimum performance improvement (speedup) that your code must achieve for each function was specified earlier in this document. Your solutions must provide at least this speedup to get a passing grade on this project.

- **Report:** Your report must briefly describe the optimizations you used and why you think they will improve the performance, and you should also provide a table of your performance results (i.e., a table showing how much better you did compared to the naive implementations). **You must provide justifications and MUST submit the report – failure to submit the report will result in a grade of zero for the entire project.** We will give you partial credit if your report is correct and your implementations do not provide the minimum speedups we require.

7   Hand In Instructions

When you have completed the lab, you will hand in (1) a report describing your solutions and (2) one file, `kernels.c`, that contains your solution.

Here is how to hand in your solution:

- You will hand in using Blackboard. Your report must be in PDF format, and must be titled Report4-lastname (For example, if your last name is Smith then your filename is Report4-smith.pdf).

- Make sure you have included your identifying information in the team struct in `kernels.c`.

- Make sure that the `rotate()` and `smooth()` functions correspond to your fastest implemnnentations, as these are the only functions that will be tested when we use our driver to grade your assignement.

- Remove any extraneous print statements.

- Create a team name of the form:

  - "*ID*" where *ID* is your ID (you can use your GW ID)

- After the handin, if you discover a mistake and want to submit a revised copy, contact the instructor.

- Submit the report in the report submission category in blackboard. Your report must briefly describe (i) the optimization techniques you used for each function, (ii) why you chose the optimizations and why you think they will improve the performance, and (iii) you should also provide a table of your performance results (i.e., a table showing how much better you did compared to the naive implementations). **Failure to submit the report will result in a grade of zero points regardless of how well your code does.**