

CENG 331

Computer Organization

Fall '2021-2022

The Performance Lab

Due Date : January 23, Sunday, 2022, 23:59

1 Objectives

This assignment deals with optimizing memory intensive code. Image processing and matrix operations offer many examples of functions that can benefit from optimization. In this homework, we will consider **Exposure Fusion** and **Gaussian Blur**. Your objective in this homework is to optimize these functions as much as possible by using the concepts you have learned in class.

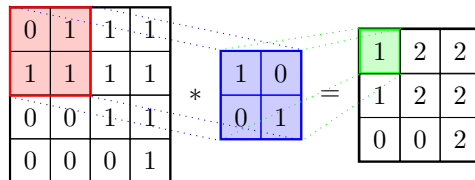
Exposure fusion merges multiple exposures into a single image. During the process, the pixels of each image are assigned weights according to their luminosity, saturation and contrast. However, you are not required to calculate the weights in this homework. The weight matrices will be provided to you. The simplified formula can be seen below:

$$R_{ij} = \sum_{k=0}^{N-1} W_{k,ij} * I_{k,ij}$$

where W_k is the k th weight matrix, I_k is the k th image matrix and R is the resulting matrix. For this homework, we will use 4 (differently exposed) images, so $N = 4$ for the above formula.

The second function, Gaussian blur, is used to reduce the noise in the images. Gaussian blur is also used when calculating weight matrices for exposure fusion. It can be applied by convolving the image with a Gaussian filter. For this homework we will use 5×5 Gaussian filter with a standard deviation of 1.

Convolution is done by calculating the sum of element-wise multiplication of the image patch and the filter, for each patch. The first two steps while convolving two matrices are represented in Figure 1.



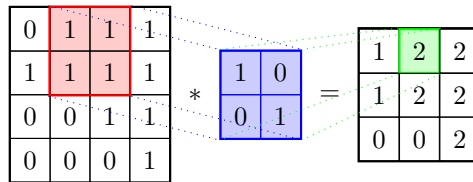


Figure 1: The first two steps of convolution

2 Specifications

Start by copying `perflab-handout.tar` to a protected directory in which you plan to do your work. Then give the command: `tar xvf perflab-handout.tar`. 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 you. **Do this right away so you don't forget.**

3 Implementation Overview

Exposure Fusion

The `fusion` function takes as two matrices `img`, `w` and returns the weighted blending of images in the destination matrix `dst`. Here is the implementation:

```
void naive_fusion(int dim, int *img, int *w, int *dst) {
    int i, j, k;
    for(k = 0; k < 4; k++)
        for(i = 0; i < dim; i++)
            for(j = 0; j < dim; j++)
                dst[i*dim+j] = w[k][i*dim+j] * img[k][i*dim+j];
}
```

Gaussian Blur

The `blur` function takes three matrices; image `img`, Gaussian filter `flt` and destination `dst`. You will only be using top left most 5×5 region of `flt` matrix for your filter. Here is the implementation:

```
void naive_blur(int dim, float *img, float *flt, float *dst) {
    int i, j, k, l;
    for(i = 0; i < dim-5+1; i++)
        for(j = 0; j < dim-5+1; j++) {
            for(k = 0; k < 5; k++)
                for(l = 0; l < 5; l++) {
                    dst[i*dim+j] = dst[i*dim+j] + img[(i+k)*dim+j+l]*flt[k*dim+l];
                }
        }
}
```

Test case	1	2	3	4	5		
Method	N	64	128	256	512	1024	Geom. Mean
Naive fusion (CPE)		7.4	7.4	7.3	7.9	9.5	1.1
Optimized fusion (CPE)		6.5	6.8	6.8	7.5	9.1	
Speedup (naive/opt)		1.1	1.1	1.1	1.1	1.0	
Method	N	64	128	256	512	1024	Geom. Mean
Naive blur (CPE)		43.3	46.0	53.3	67.0	69.4	1.9
Optimized blur (CPE)		22.9	24.3	31.6	31.8	33.2	
Speedup (naive/opt)		1.9	1.9	1.7	2.1	2.1	

Table 1: CPEs and Ratios for Optimized vs. Naive Implementations

}

As you can see from Figure 1, the resultant image is smaller than the original one. For simplicity, you should be filling the destination matrix `dst` starting from top left pixel and leave the rest untouched. `dst` is a zero matrix, initially.

Performance Measures

Our main performance measure is *CPE* or *Cycles per Element*. If a function takes C cycles to run for an image of size $N \times N$, the CPE value is C/N^2 . Table 1 summarizes the performance of the naive implementations shown above and compares it against an optimized implementation. Performance is shown for 5 different values of N . All measurements were made on the department computers (ineks).

The ratios (speed-ups) of the optimized implementation over the naive one will constitute a *score* of your implementation. To summarize the overall effect over different values of N , we will compute the *geometric mean* of the results for these 5 values. That is, if the measured speed-ups for $N = \{64, 128, 256, 512, 1024\}$ are $R_{64}, R_{128}, R_{256}, R_{512}$, and R_{1024} then we compute the overall performance as

$$R = \sqrt[5]{R_{64} \times R_{128} \times R_{256} \times R_{512} \times R_{1024}}$$

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 the 5 values shown in Table 1.

For both functions, we will use square grayscale images.

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 `fusion` and `blur` 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 functions:

```
void register_fusion_functions() {
    add_fusion_function(&fusion, fusion_descr);
}
void register_blur_functions() {
    add_blur_function(&blur, blur_descr);
}
```

This function contains one or more calls to `add_fusion_function` and `add_blur_function`. In one of the examples above, `add_fusion_function` registers the function `fusion` along with a string `fusion_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. Gaussian blur works the same way.

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 `fusion()` and `blur()` functions are run. This is the mode we will use when grading your solution.
- *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 `fusion()` and `blur()` functions (*autograder mode*).

`-f <funcfile>` : Execute only those versions specified in `<funcfile>` (*file mode*).

`-d <dumpfile>` : Dump the names of all versions to a dump file called `<dumpfile>`, 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.

Team Information

Important: Before you start, you should fill in the struct in `kernels.c` with information about your team (team name, student names, student ids).

5 Assignment Details

Optimizing Exposure Fusion (30 points)

In this part, you will optimize `fusion` to achieve as low a CPE as possible. You should compile `driver` and then run it with the appropriate arguments to test your implementations.

For example, running `driver` with the supplied naive version (for `fusion`) generates the output shown below:

```
unix> ./driver
Teamname: Team
Member 1: Student Name
ID 1: eXXXXXXX

fusion: Version = naive_fusion: Naive baseline exposure fusion:
Dim          64      128      256      512      1024      Mean
Your CPEs    7.4      7.5      7.4      8.1      9.7
Baseline CPEs 7.4      7.4      7.3      7.9      9.5
Speedup      1.0      1.0      1.0      1.0      1.0      1.0
```

Optimizing Gaussian Blur (70 points)

In this part, you will optimize `blur` to achieve as low a CPE as possible. You should compile `driver` and then run it with the appropriate arguments to test your implementations.

For example, running `driver` with the supplied naive version (for `blur`) generates the output shown below:

```
unix> ./driver
Teamname: Team
Member 1: Student Name
ID 1: eXXXXXXX

blur: Version = naive_blur The naive baseline version of Gaussian blur:
Dim          64      128      256      512      1024      Mean
Your CPEs    43.2     46.0     54.0     67.0     69.9
Baseline CPEs 43.3     46.0     53.3     67.0     69.4
Speedup      1.0      1.0      1.0      1.0      1.0      1.0
```

Some advice. Look at the assembly code generated for the `fusion` and `blur`. Focus on optimizing the inner loop (the code that gets repeatedly executed in a loop) using the optimization tricks covered in class.

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 this file.

Evaluation

- **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.
- Note that you should only modify `dst` pointer. If you modify input pointers or exceed their limits, you will not get any credit.
- **Exposure Fusion:** You will get 30 points if your implementation is correct and achieve a mean speed-up of 1.1. There is no scaling in this function.
- **Gaussian Blur:** You will get 40 points for implementation of `blur` if it is correct and achieve mean speed-up threshold of 1.9. The team that achieves the highest speed-up will get 70 points. Other grades will be scaled between 40 and 70 according to your speed-up. You will not get any partial credit for a correct implementation that does below the threshold.
NOTE: In order to have an idea of what will be your grade, you should know about speed-ups of other students. Therefore, sharing your highest speed-ups is highly recommended.
- Since there might be changes of performance regarding to CPU status, test the same code many times and take only the best into consideration. When your codes are evaluated, your codes will be tested in a closed environment many times and only your best speed-up of functions will be taken into account.
- **Optional:** In this assignment, you have an option to create a team up to three people. However, you can also do it alone.

Submission

Submissions will be done via ODTUClass. You can only submit `kernels.c` function. Therefore, make sure all of your changes are only on this file. Any member of the team can submit the file. Do **NOT** submit the same file by different members of the team. One submission is enough.