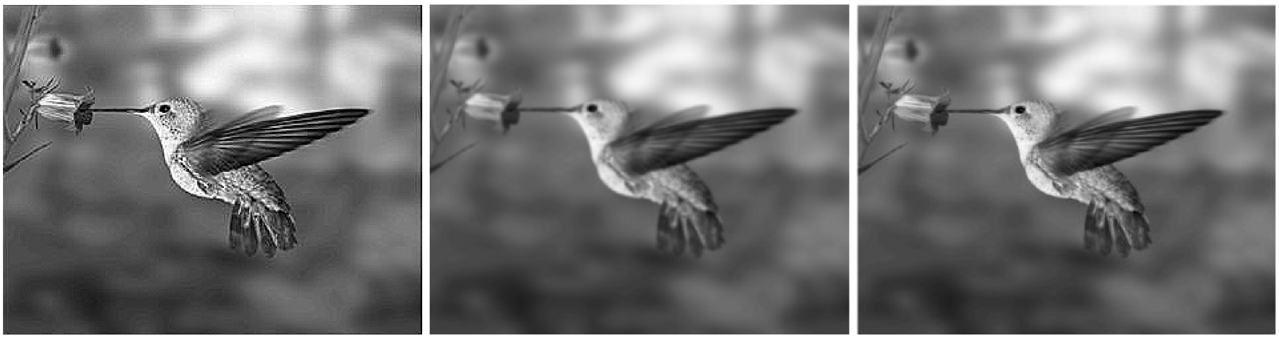
Parallel Programming
Assignment 3

Mahmood Naseer
Computer Science Department

December 18, 2023



(a) Test kernel K3_Sharpen.txt (b) Test kernel K1_Blur.txt (c) Original Image

Figure 1: Comparison of Images

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

(a) K2_Edge_Detection

$$\begin{bmatrix} 0.0625 & 0.125 & 0.0625 \\ 0.125 & 0.25 & 0.125 \\ 0.0625 & 0.125 & 0.0625 \end{bmatrix}$$

(b) K1_Blur.txt

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

(c) K3_Sharpen.txt

How the Code Works

When running the code, you must provide the compiled file with the arguments. For parallel code (papply_filter.cpp):

```
$ ./a.out <image> <kernel> <number of threads>
```

For serial code (apply_filter.cpp), the command is similar but without specifying the number of threads:

```
$ ./a.out <image> <kernel>
```

Implementation

The code in both files, apply_filter.cpp and papply_filter.cpp, is similar. However, in the pthread code (papply_filter.cpp), organization and synchronization were implemented.

Let's start with the distribution of work:

```
data = ((image_x * image_y) / thread_size);
start = my_rank * data;
stop = my_rank * data + data;
// 'data' represents the number of pixels each thread will perform Convolution
   calculations on.
```

The data in the file can be treated as a vector. To access the data, treat it as a 2D array:

```
// 'j' is the y access or the row, 'i' is the x access or column.
// '_image->size()' is the same as image_x * image_y, the total number of pixels in the
   vector.
loc_image = ((j * image_x) + i + k) % _image->size();
```

Using this equation, Convolution can be calculated by taking the sum of the multiplication of each pixel in both matrices. After processing all pixels, a semaphore must be implemented to synchronize the order of saving threads to the file. Each thread has its own buffer of calculated data, and the semaphore allows saving processor resources by putting threads to sleep and waking them up when their job is done. All threads save their work to the file.

Finally, the allocated memory resources must be cleaned up.

Digital image processing & pthread

The code was tested on two pictures:

1. The provided picture (*oimage.txt*) 253x320.
2. A second high-quality picture converted to grayscale (*image_huge.txt*) 3403x5266.
3. Used Kernel (*K2_Edge_Detection.txt*)

image_huge.txt Time 866.914ms

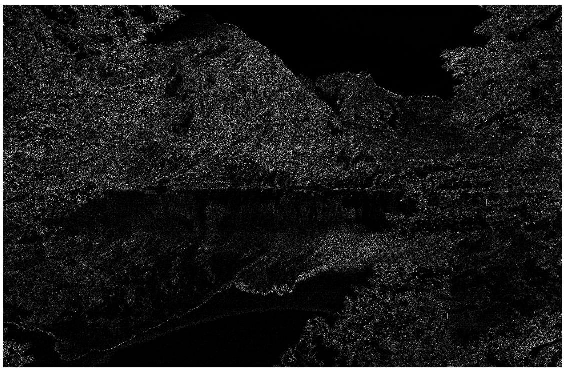


Figure 3: output code (grayscale) image_huge.txt 3403x5266.

Thread	Avg(ms)	Max(ms)	Min(ms)
4	877.710	914.540	839.463
8	536.950	563.100	512.008
16	439.010	485.528	360.294

Table 1: papply_filter.cpp Time 866.914ms

oimage.txt Time 11.3606ms

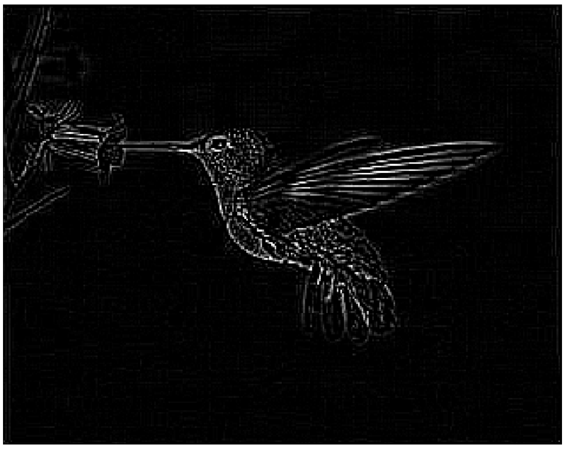


Figure 4: output code (grayscale) oimage.txt 253x320.

Thread	Avg(ms)	Max(ms)	Min(ms)
4	4.1768175	4.61135	3.58399
8	3.3453833	3.76275	2.83007
16	1.87293	2.89754	1.03466

Table 2: papply_filter.cpp