Matrix Convolution for Image Processing Using Parallel Computing

Yacine Sahli, E10715005

2019

Objective:

Matrix convolution are often used in image processing but necessite quite a large amount of computation due to the computation needed for every pixel on the image/frame.

My objective is to make image convolution for large images done in a blink of an eye.

Methods and Algorithms

The basic algorithm I used is as follow:

```
for each image row in input image:
    for each pixel in image row:
        set accumulator to zero
        for each kernel row in kernel:
                for each element in kernel row:
                    if element position  corresponding* to pixel position then
                            multiply element value  corresponding* to pixel value
                            add result to accumulator
                        endif
            set output image pixel to accumulator
```

To achieve a good serial code, I used methods such as loop unrolling and tried many different data layout to optimize the cache access.

I also used a technique called kernel separation which is very useful for large kernel as it exponentially reduce the number of operations as the kernel get larger.

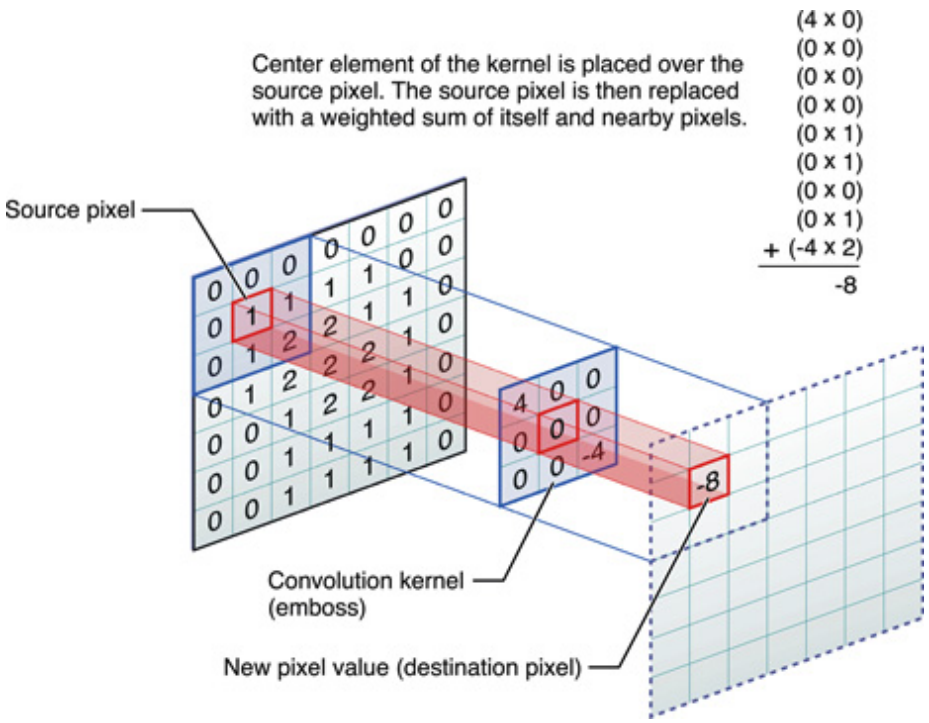


Figure 1: Image Convolution

Sample calculations

for a 1920 * 1080 px image and a : 3*3 convolution matrix* → *18 662 400 operations*
*9*9 convolution matrix → 167 961 600 operations

Flowchart

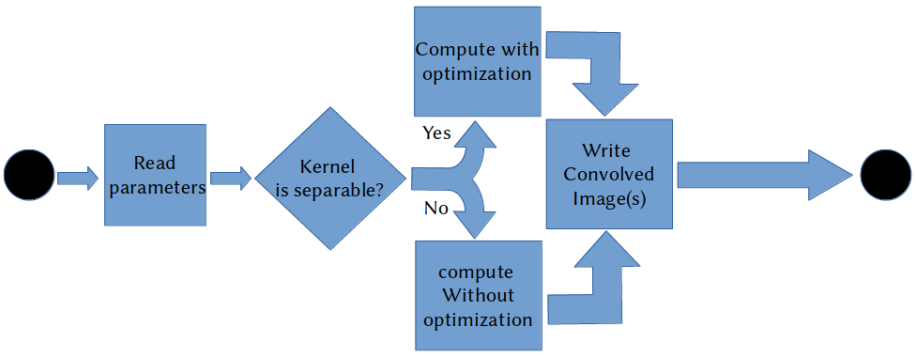


Figure 2: Flowchart

SIMD

This is the lowest level parallelization technique and I let the compiler handle it by itself using -O3 flag.

OpenMP

I used OpenMP to parallelize my program trying a lot of different combination of parameters and preprocessor directive.

The scheduling used is static as it is the one with the smallest overhead and every chunk is balanced.

I let OpenMP decide the chunk size as fiddling with it did not improve the performance.

I put together parallel region to avoid the thread allocation/parking overhead in the 1dConvolution.

Application

The simplest application could be a program applying predetermined or custom filter to an image/images in a blink of an eye, like on a mobile application.

More complicated would be to apply filter on a video frame by frame, which would necessite a very quick image processing if we want to make it appear in live directly using a camera.

A lot of different filter can be applied with convolution matrix, for example: Gaussian blur, unblur, Sharpening, unsharpening, embossing, edge detection ... .

Serial implementation

Program's inputs

```
OMP_NUM_THREADS=1 ./main.exe input kernel output
```

None of the parameters are optional.

If OMP_NUM_THREADS=1 is not specified, the program will run in parallel mode with openMP.

- input can be a .jpeg .jpg .png image or a folder containing multiples images in the same format.
- kernel as to be a file containing a kernel with the dimensions of the kernel in the first line, for example a box blur kernel file

Figure 3: Kernel example

- output can be a .jpeg .jpg .png image or a folder.

Execution time

image:1620x1080 kernel:3x3 convolution time:0.080s

image:1620x1080 kernel:5x5 convolution time:0.087s

image:1620x1080 kernel:7x7 convolution time:0.107

image:1620x1080 kernel:9x9 convolution time:0.110

image:9865x3120 kernel:3x3 convolution time:3.409

image:9865x3120 kernel:5x5 convolution time:3.757

image:9865x3120 kernel:7x7 convolution time:4.027

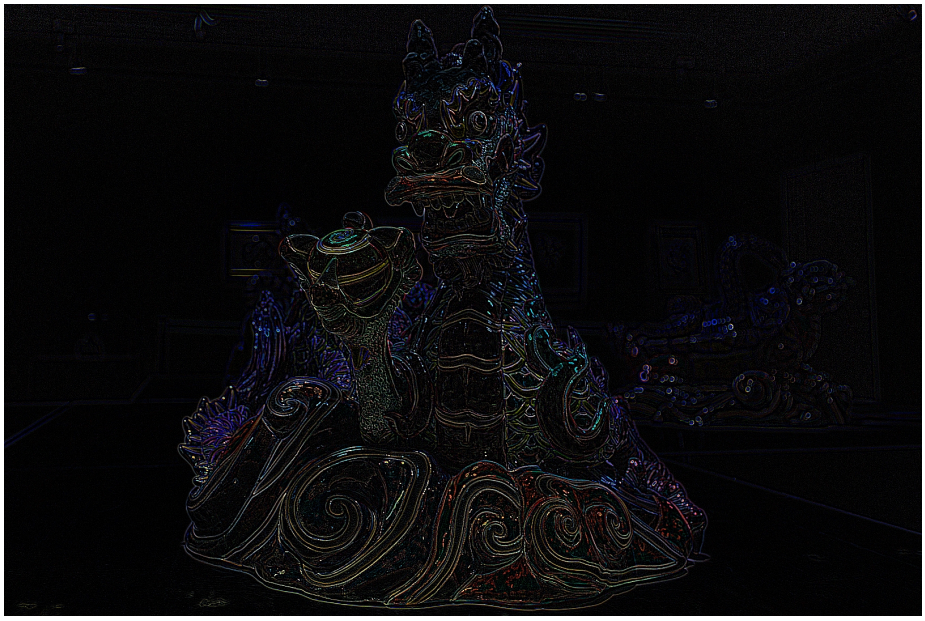
image:9865x3120 kernel:9x9 convolution time:4.071

Output

Here is the same image with 3 differents kernels used for testing my program.

Original dragon picture

Same picture after Gaussian blur 5x5



Same picture after High pass 3x3 filter

Same picture after emboss 3x3 filter

Validations

See Output, resulting images are as expected by the filter, blurred, embossed or high passed.

Parallel implementation

Makefile

The program and his Makefile is on the group cluster at `/e10715005/final_project/ImageConvolutionOpenMP`

`make test` will run a quick test.

`make valgrind` will run the same test with valgrind.

`make speed` will run multiple small test.

`make folder` will run a test on a folder.

`make stats` will generate a serie of statistics file used to make the graph plotting in this report.

Program's inputs

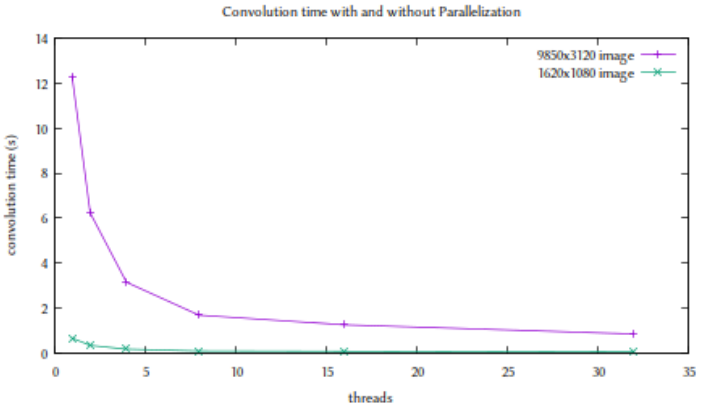
```
./main.exe input kernel output
```

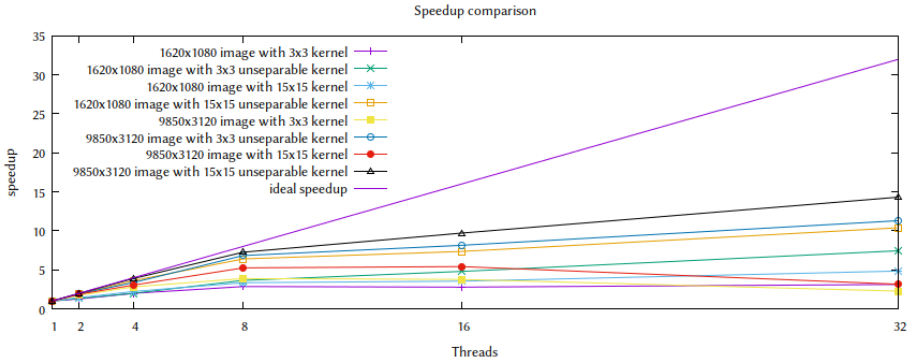
None of the parameters are optional.

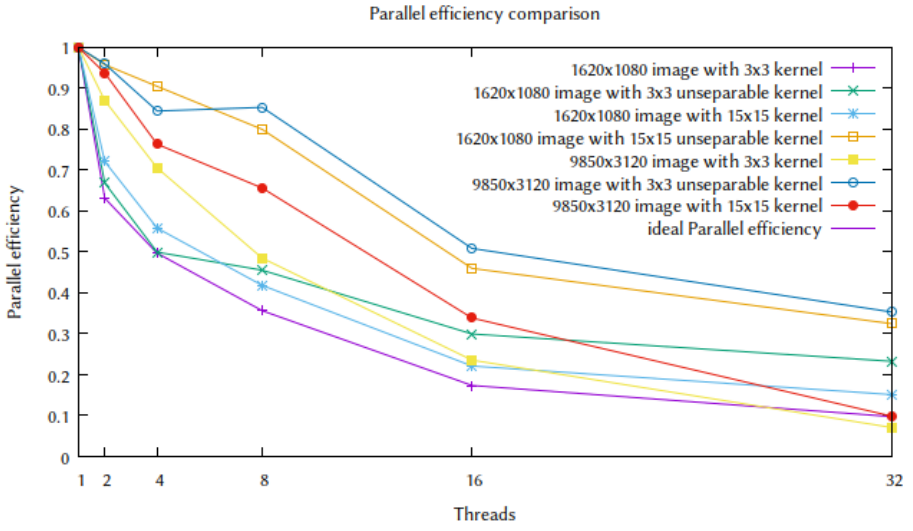
- input can be a .jpeg .jpg .png image or a folder containing multiples images in the same format.
- kernel as to be a file containing a kernel with the dimensions of the kernel in the first line, for example a box blur kernel file

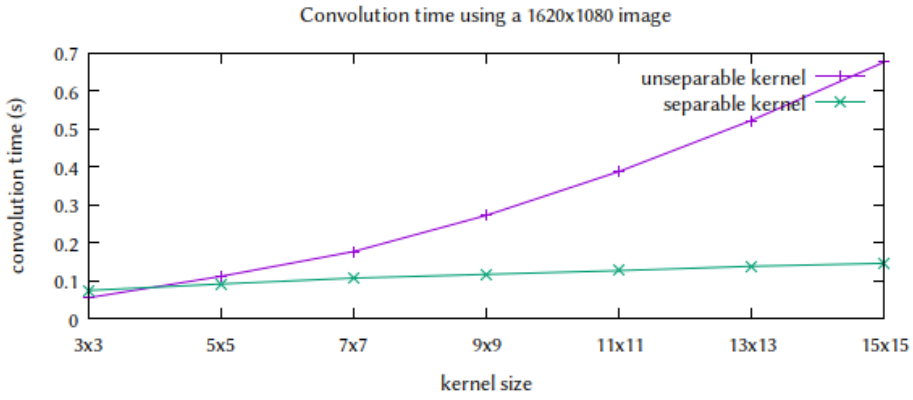
By default openmp will use the maximum number of thread available but you can also set the maximum number at runtime by adding `OMP_NUM_THREADS=N` before executing the program, with N the maximum number of thread.

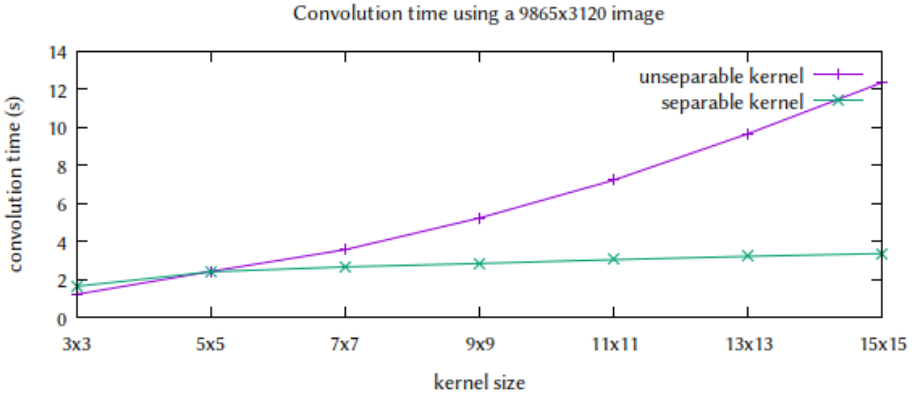
Performance






Parallel efficiency comparison
Parallel efficiency
Threads
1620x1080 image with 3x3 kernel
1620x1080 image with 3x3 unseparable kernel
1620x1080 image with 15x15 kernel
1620x1080 image with 15x15 unseparable kernel
9850x3120 image with 3x3 kernel
9850x3120 image with 3x3 unseparable kernel
9850x3120 image with 15x15 kernel
ideal Parallel efficiency
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
1
2
4
8
16
32


Convolution time using a 1620x1080 image
convolution time (s)
kernel size
unseparable kernel
separable kernel
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
3x3
5x5
7x7
9x9
11x11
13x13
15x15



If we have large number of core, preferable to not use separable kernel as speedup is better with the not separable version.

We can see that the kernel separation is very powerful for large kernel as the convolution time will grow exponentially as the kernel size grow and the convolution time will grow linearly with the kernel separation.

Implementation

Everything is in the **`/home/e10715505/final_project/ImageConvolutionOpenMP`** folder

Validations

I ran a test convolving the same dragon.jpg with box_blur3 with 1 and 4 threads and stored each of the resulting file in tmp1.jpg and tmp2.jpg.

Then I ran **`diff tmp1.jpg tmp2.jpg`** which returned nothing so the 2 files are the same with 1 or multiple threads.

I also ran **`diff tmp1.jpg dragon.jpg`** to test the blurred image with the original one and diff returned that the two binary file are different, which is normal because tmp1.jpg has been blurred.

```
e10715050@cluster2 ~/e10715005/final_project/ImageConvolutionOpenMP $ OMP_NUM_THREADS=1 ./main.exe images/dragon.jpg kernels/box_blur3 tmp1.jpg
1620x1080 3x3 1 0.0982336 0.122991 0.017676
e10715050@cluster2 ~/e10715005/final_project/ImageConvolutionOpenMP $ OMP_NUM_THREADS=4 ./main.exe images/dragon.jpg kernels/box_blur3 tmp2.jpg
1620x1080 3x3 4 0.0988733 0.0624255 0.020923
e10715050@cluster2 ~/e10715005/final_project/ImageConvolutionOpenMP $ diff tmp1.jpg tmp2.jpg
e10715050@cluster2 ~/e10715005/final_project/ImageConvolutionOpenMP $ diff tmp1.jpg images/dragon.jpg
Binary files tmp1.jpg and images/dragon.jpg differ
e10715050@cluster2 ~/e10715005/final_project/ImageConvolutionOpenMP $
```

References

https://jeanvitor.com/convolution-parallel-algorithm-python/Convolution parallel algorithm using python https://www.ijsr.net/archive/v2i7/SUpTUk9GRjIwMTMzMTg=.pdf
Matrix convolution using Parallel Programming https://www.researchgate.net/publication/321325064_2D_Imag
https://www.ijsr.net/archive/v2i7/SUpTUk9GRjIwMTMzMTg=.pdf