Project Report

Manoj Bhadu-170010036 Rishi Agarwal 170010047
Prince Sharma-170010017 Piyush Onkar-160050012
Kaustubh Tendolker- 170110020

October 2020

Contents

1 Abstract 1

2 Serial Implementation 1
2.1 Gaussian Blurring . . . 1
2.2 Sobel Operation . . . 2

3 Parallel Implementation 4
3.1 Gaussian Blurring . . . 4
3.2 Sobel Operator . . . 5

4 Results and Time Analysis 6
4.1 Processing Images . . . 6
4.2 Timing Results . . . 7

5 Conclusion 8

1 Abstract

Images are essentially matrices of real numbers describing the intensity value at those pixels. Hence, image processing algorithms are essentially calculations involving matrix operations that can be parallelized. In this porject we will focous on parallelizing Guassian blurring and Edge detection algorithm in pyhon using PyMP which is based on OpenMP.

2 Serial Implementation

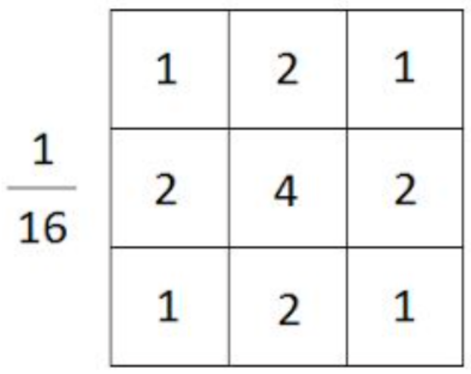
2.1 Gaussian Blurring

In image processing, a Gaussian blur (also known as Gaussian smoothing) is the result of blurring an image by a Gaussian function (named after mathematician

and scientist Carl Friedrich Gauss).
It is a widely used effect in graphics software, typically to reduce image noise and reduce detail. The visual effect of this blurring technique is a smooth blur resembling that of viewing the image through a translucent screen, distinctly different from the bokeh effect produced by an out-of-focus lens or the shadow of an object under usual illumination.

The Gaussian blur is a type of image-blurring filter that uses a Gaussian function (which also expresses the normal distribution in statistics) for calculating the transformation to apply to each pixel in the image. The formula of a Gaussian function in one dimension is
$G(x) = \frac{1}{\sqrt{2\pi\sigma}} e^{\frac{x^2}{2\sigma^2}}$
Similarly In two dimensions, it is the product of two such Gaussian functions. Values generated from this distribution is convoluted over entire image. The 2d-convolution operation is performed between the image and a small matrix (we take 3x3 in our case). Where onvolution process basically, calculates for each pixel, the new value by adding weighted values of the neighbouring pixels.

We are using the kernel that is used for blurring is as follows:



$\frac{1}{16}$

1	2	1
2	4	2
1	2	1

Figure 1: Kernel used for Gaussian Blur

2.2 Sobel Operation

The Sobel operator, sometimes called the Sobel–Feldman operator or Sobel filter, is used in image processing and computer vision, particularly within edge detection algorithms where it creates an image emphasising edges. It is named after Irwin Sobel and Gary Feldman. It is a discrete differentiation operator, computing an approximation of the gradient of the image intensity function. At each point in the image, the result of the Sobel–Feldman operator is either the corresponding gradient vector or the norm of this vector. The Sobel–Feldman

Algorithm 1: Gaussian Blur serial

```
1:  Input: Image Matrix, Kernel
2:  for $i = 1, \cdots$ rows in image do
3:      for $j = 1, \cdots$ columns in image do
4:          set accumulator to zero
5:          for $k = 1, \cdots$ rows in kernel do
6:              for $l = 1, \cdots$ columns in kernel do
7:                  if element position corresponding* to pixel position then multiply
                    element value corresponding* to pixel value
8:                  add result to accumulator endif
9:              end for
10:         end for
11:         set output image pixel to accumulato
12:     end for
13: end for
```

operator is based on convolving the image with a small, separable, and integer-valued filter in the horizontal and vertical directions and is therefore relatively inexpensive in terms of computations

The Sobel operator is used to detect two kinds of edges namely, horizontal and vertical edges. The Sobel operator when applied to an image basically uses the convolution process to obtain the edges

It works on the principle that a change in the gradient of intensity of pixels almost certainly will be an edge of the image, i.e. the areas where there is a significant change in the intensity of the pixels marks a different object in the image.

The horizontal edges are identified using the horizontal mask as shown below, where A represents the image matrix

$$\mathbf{G}_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} * \mathbf{A}$$

Figure 2: Horizontal Derivative

Algorithm 2: Sobel Operation Serial Implementation

```
1: Input: Image Matrix, horizontal and vertical Kernel
2: for $i = 1, \cdots$ pixel in image do
3:     Compute the convolved value of the pixel using the horizontal Sobel kernel
4: end for
5: for $i = 1, \cdots$ pixel in image do
6:     Compute the convolved value of the pixel using the vertical Sobel kernel
7: end for
8: for $i = 1, \cdots$ pixel in image do
9:     Compute the resultant gradient approximation
10: end for
```

The vertical edges are identified using the horizontal mask as shown below, where A represents the image matrix
Now, to obtain the final image that combines both horizontal and vertical

$$\mathbf{G}_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * \mathbf{A}$$

Figure 3: Vertical Derivative

images, the following equation is applied to the Gx and Gy matrices.
$\mathbf{G} = \sqrt{\mathbf{G_x^2} + \mathbf{G_y^2}}$

3 Parallel Implementation

3.1 Gaussian Blurring

Convolution operation involves pixel wise computation for the input image. Computation for each pixel are independent of each other, i.e they are embarrassingly parallel and hence the computation on each pixel can be performed simultaneously. For achieving this parallelism, PyMP for Python were used which use thread-level parallelism.
We set threads to different values and did time analysis.
Part from algorithm 1 parallelized

```
for i in p1.range(1,l+1):
    for j in p2.range(1,b+1):
        do computations
```

This nested for loops are parallelized to compute new pixel values simultaneously as follows:

```
with pymp.Parallel(2) as p1:
    with pymp.Parallel(2) as p2:
        for i in p1.range(1,l+1):
            for j in p2.range(1,b+1):
                do computations
```

3.2 Sobel Operator

Similar to Gaussian blurring the convoluation is parallized and the final calculation of derivative is also done using parallelization.
Parts of the code parallelized were:

```
with pymp.Parallel(2) as p1:
    with pymp.Parallel(2) as p2:
        for i in p1.range(1,l+1):
            for j in p2.range(1,b+1):
                do computations
```

above mentioned code is used for parallizing both horizontal and vertical convolution and.
After this the third for loop in algorithm 2 is also parallalized using same commands and method for calculation of **G**

4 Results and Time Analysis

4.1 Processing Images

Figure 4 and Figure 5 depicts the images we have obtained after applying the Gaussian blur and Sobel operation repectively.

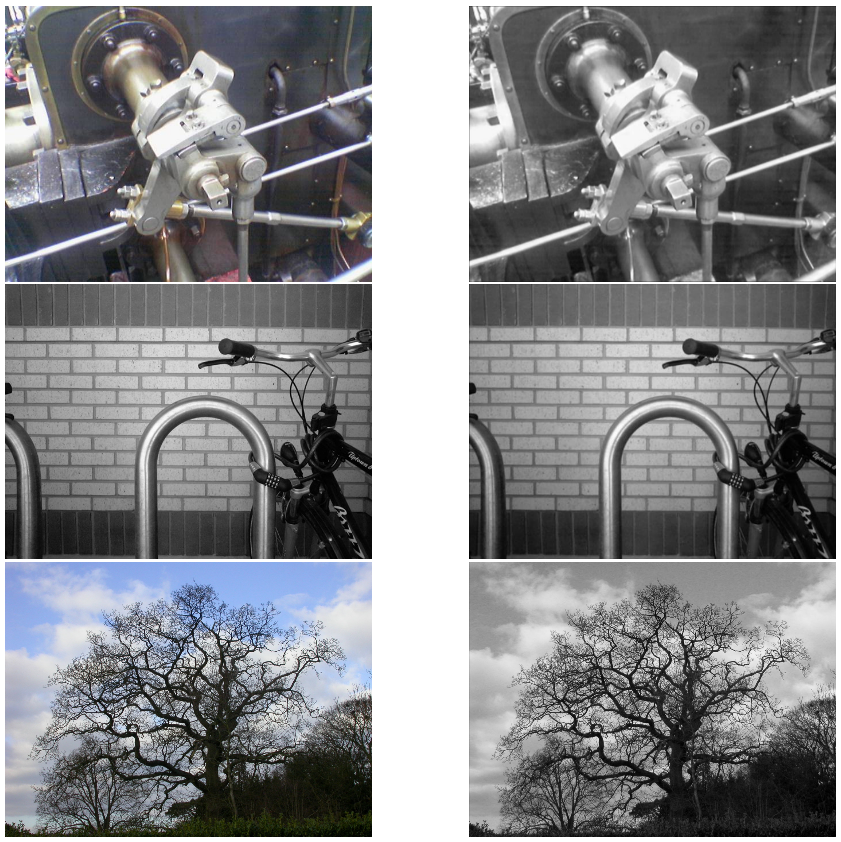


Figure 4: Images Before(Left) and After(Right) Applying Gaussian Blur

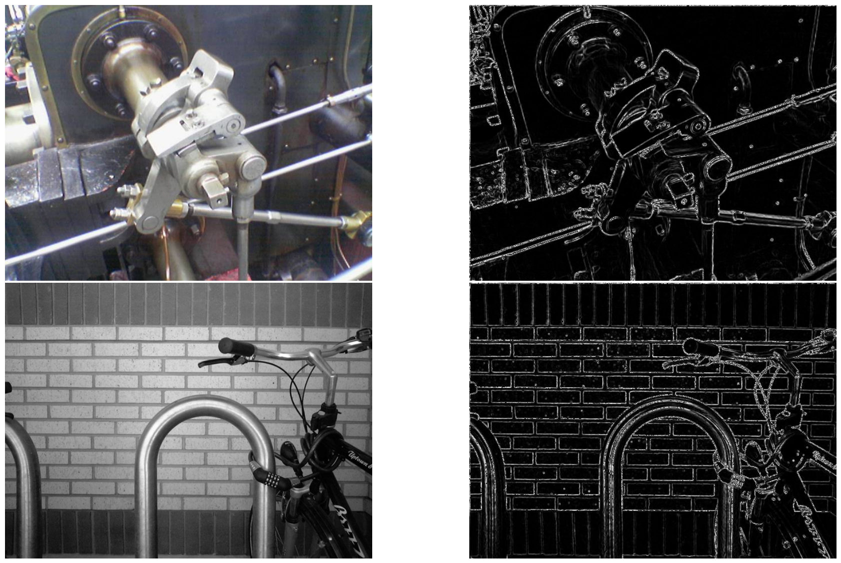


Figure 5: Images Before(Left) and After(Right) Applying Sobel Operator

4.2 Timing Results

- **Gaussian Blurring**

Sizes	2 Threads	4 Threads	6 Threads	8 Threads	Serial
480 x 640	1.01	0.99	1.12	1.15	4.19
800 x 1000	3.67	2.86	3.12	3.21	12.34
1200 x 1400	8.34	7.94	8.16	9.56	22.56
1704 x 2272	13.90	11.75	12.90	13.25	40.61
3610 x 2488	26.98	22.74	24.93	23.56	74.04
4032 x 2268	33.82	26.56	28.76	30.41	104.76
3999 x 3000	34.42	29.25	30.67	32.56	106.97

- **Sobel Operation**

Sizes	2 Threads	4 Threads	6 Threads	8 Threads	Serial
480 x 640	4.07	10	13	786	6.27
800 x 1000	6.78	5.89	6.32	6.98	15.45
1200 x 1400	11.84	10.53	11.01	12.09	22.43
1704 x 2272	31.66	27.36	28.67	29.45	90.92
3610 x 2488	75.86	65.51	68.18	69.34	226.45
4032 x 2268	79.09	70.94	72.78	76.19	224.72
3999 x 3000	97.22	82.56	89.37	86.56	267.12

5 Conclusion

- As we can see from the results mentioned in above section the timing is improved around 1.5x to 2x and even 3x for higher pixel resolution of images going from serial to thread parallelization for both Gaussian blur as well as Sobel operation
- As size of images increases the improvement is more clearly visible because as the size increases the pixel number increases hence the computation that can be parallelized increases.
- The Time taken in going from serial to 2,4 threads is clearly increases but after 4 threads the time taken is fluctuating due to our machine configuration(as machine used have 4 cores)