Joshua Caudill

CS6068

Assignment #3

Table of Contents

Table of Figures

Introduction

The goal of this assignment was to achieve fast blurring of images. A noisy image was blurred first by using a square stencil and calculating the mean intensity values for each color channel over the stencil for each pixel. The noisy image was blurred next by using Gaussian blurring, which was defined as exponentially weighted averaging using a square stencil at every pixel. The speedup was calculated using the sequential execution time and the parallel execution time for the Gaussian blurring technique using different filter sizes. The Numba @jit decorator was added to functions to parallelize execution.

Tools

- gaussian_blur.py
- time, matplotlib.pyplot, numpy, numba.jit, PIL.Image
- Personal Desktop
    - AMD Ryzen 7 7700X 8-Core Processor 4.50 GHz
    - NVIDIA GeForce RTX 4080 SUPER
    - 32.0 GB RAM
- Python 3.12.3

Code

```
"""
Reads in a noisy image, applies a Gaussian blurring filter to it, and saves the resulting image.
"""


import time


import matplotlib.pyplot as plt
import numpy as np
from numba import jit
from PIL import Image


KERNEL_LENGTH = 5
SIGMA = 1


@jit(nopython=True)
def generate_kernel():
    """
    Generates Gaussian kernel with length and sigma.
    """
    ax = np.linspace(
        -(KERNEL_LENGTH - 1) / 2.0, (KERNEL_LENGTH - 1) / 2.0, KERNEL_LENGTH
    )
```

```
    gauss = np.exp(-0.5 * np.square(ax) / np.square(SIGMA))
    k = np.outer(gauss, gauss)
    return k / np.sum(k)  # Return the normalized kernel.


@jit(nopython=True)
def blur(in_img, out_img, k):
    """
    Applies the Gaussian kernel to the noisy image and saves the resulting image.
    """
    for c in range(3):
        for x in range(in_img.shape[1]):
            for y in range(in_img.shape[0]):
                val = 0
                for i in range(-(KERNEL_LENGTH // 2), ((KERNEL_LENGTH // 2) + 1)):
                    for j in range(-(KERNEL_LENGTH // 2), ((KERNEL_LENGTH // 2) + 1)):
                        if (
                            (x + i < in_img.shape[1])
                            and (x + i >= 0)
                            and (y + j < in_img.shape[0])
                            and (y + j >= 0)
                        ):
                            val += int(in_img[y + j, x + i, c]) * k[i, j]
                out_img[y, x, c] = val


if __name__ == "__main__":
    img = np.array(Image.open("noisy1.jpg"))  # Open the noisy image.
    imgblur = img.copy()  # Save a copy of the noisy image.
    start = time.time()  # Get the start time.
    kernel = generate_kernel()  # Generate the kernel.
    blur(img, imgblur, kernel)  # Blur the image.
    end = time.time()  # Get the end time.
    print(f"Elapsed = {(end - start)}")  # Print the elapsed time.

    # Display and save blurred image.
    fig = plt.figure()
```

```
axes = fig.add_subplot(1, 2, 1)

imgplot = plt.imshow(img)

axes.set_title("Before")

axes = fig.add_subplot(1, 2, 2)

imgplot = plt.imshow(imgblur)

axes.set_title("After")

img2 = Image.fromarray(imgblur)

img2.save("blurred.jpg")

plt.show()
```

Results

The mean intensity value blurring code from the textbook was executed first. The Python script, gaussian_blur.py, was then executed in both sequential and parallel fashion with the following filter sizes: 5, 7, and 9. The Python script was executed in a sequential fashion with the @jit decorators commented, and it was executed in a parallel fashion with the @jit decorators uncommented. The speedups shown below were calculated. The screenshots shown below were taken from my personal desktop. Numba is especially good at generating efficient compiled code for scripts that use NumPy, which explained the speedups that were achieved.

- Speedup [Length 5] $=$ 46.40 sec / 1.369 sec $=$ 33.89
- Speedup [Length 7] $=$ 89.51 sec / 1.608 sec $=$ 55.67
- Speedup [Length 9] $=$ 144.7 sec / 1.730 sec $=$ 83.64

Figure 1: Mean Intensity Blur

Figure 2: Gaussian Blur, Length 5

Figure 3: Gaussian Blur, Length 7

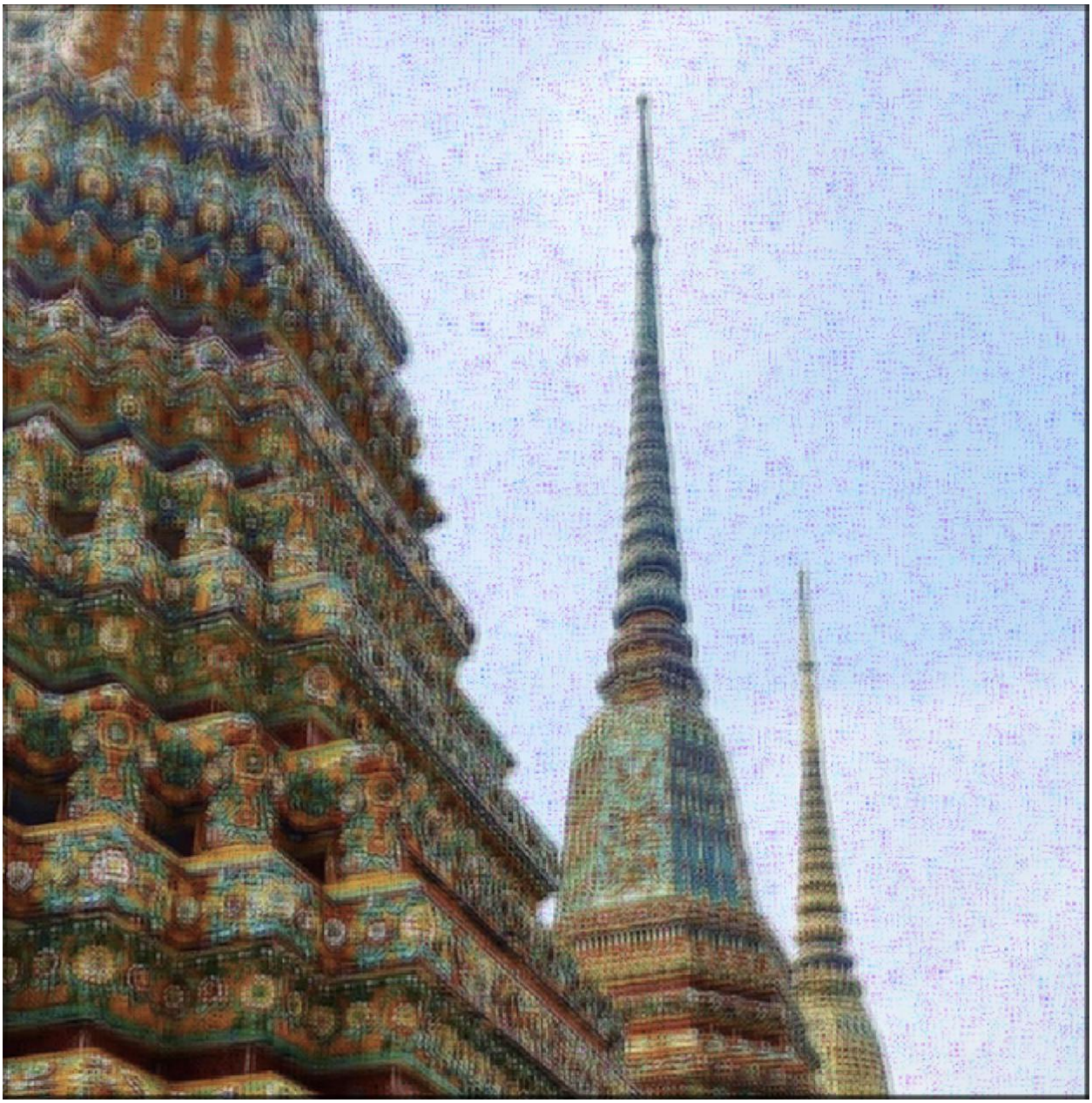

Figure 4: Gaussian Blur, Length 9

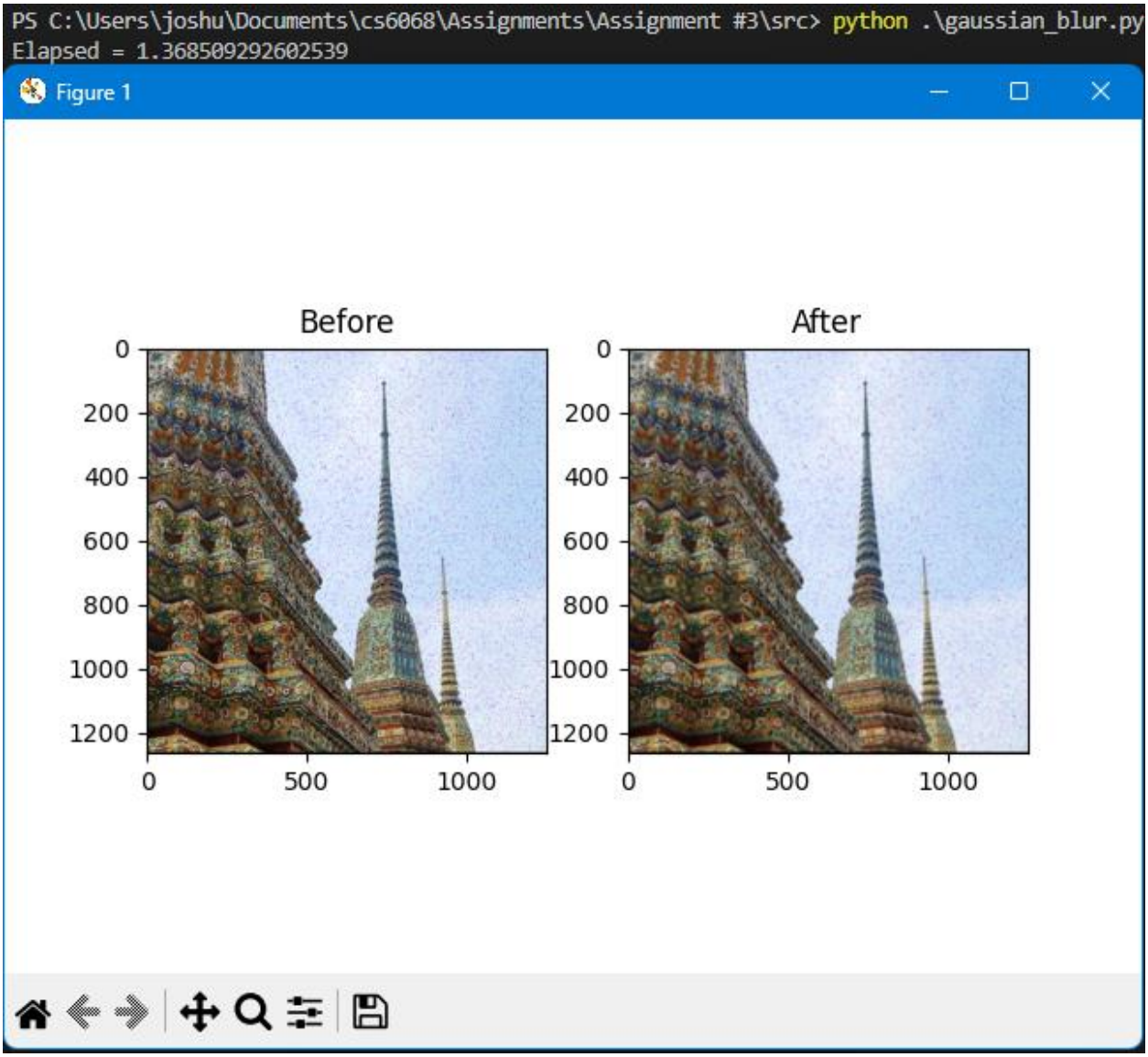

Figure 5: Gaussian Blur, Parallel, Length 5

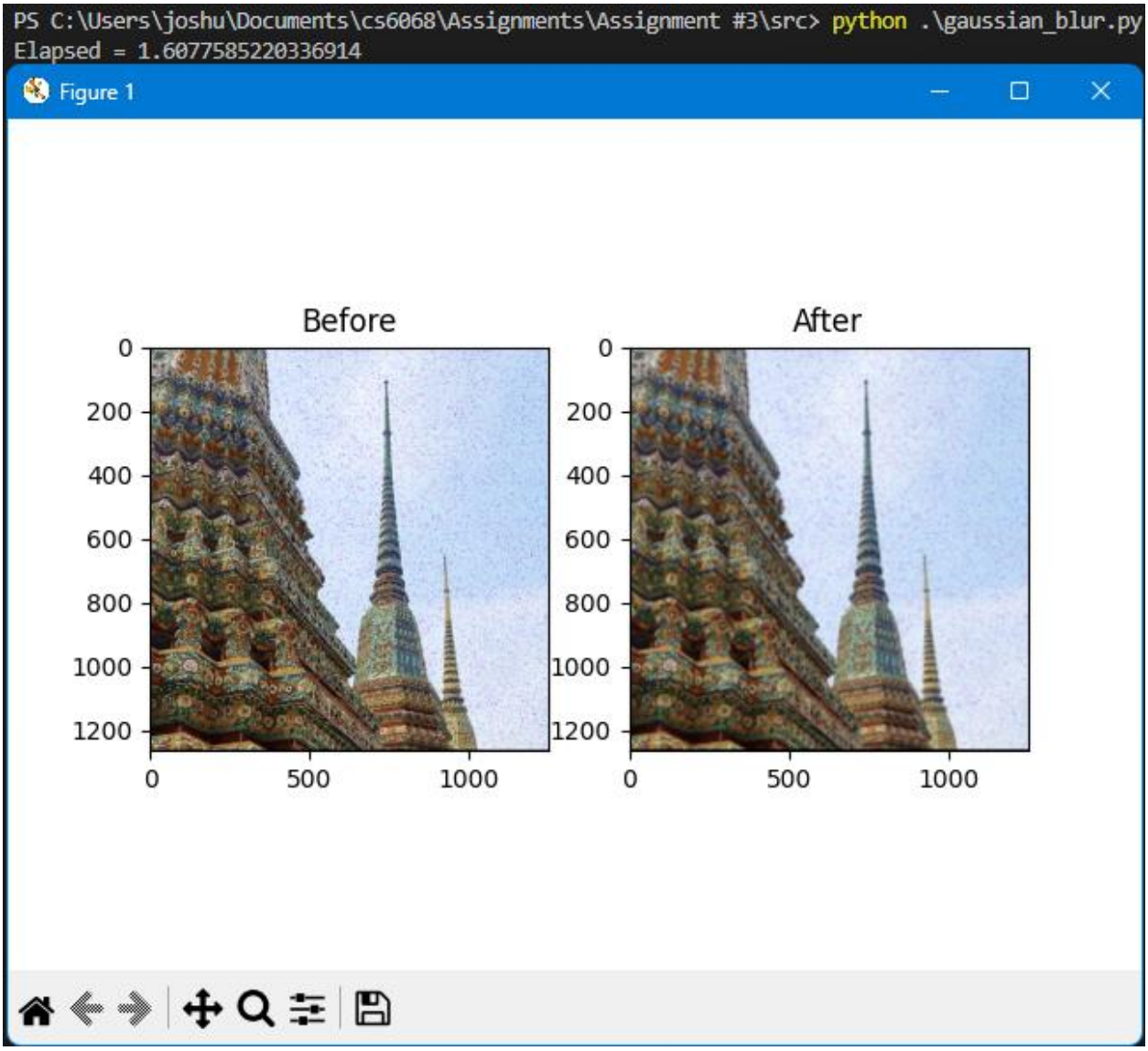

Figure 6: Gaussian Blur, Parallel, Length 7

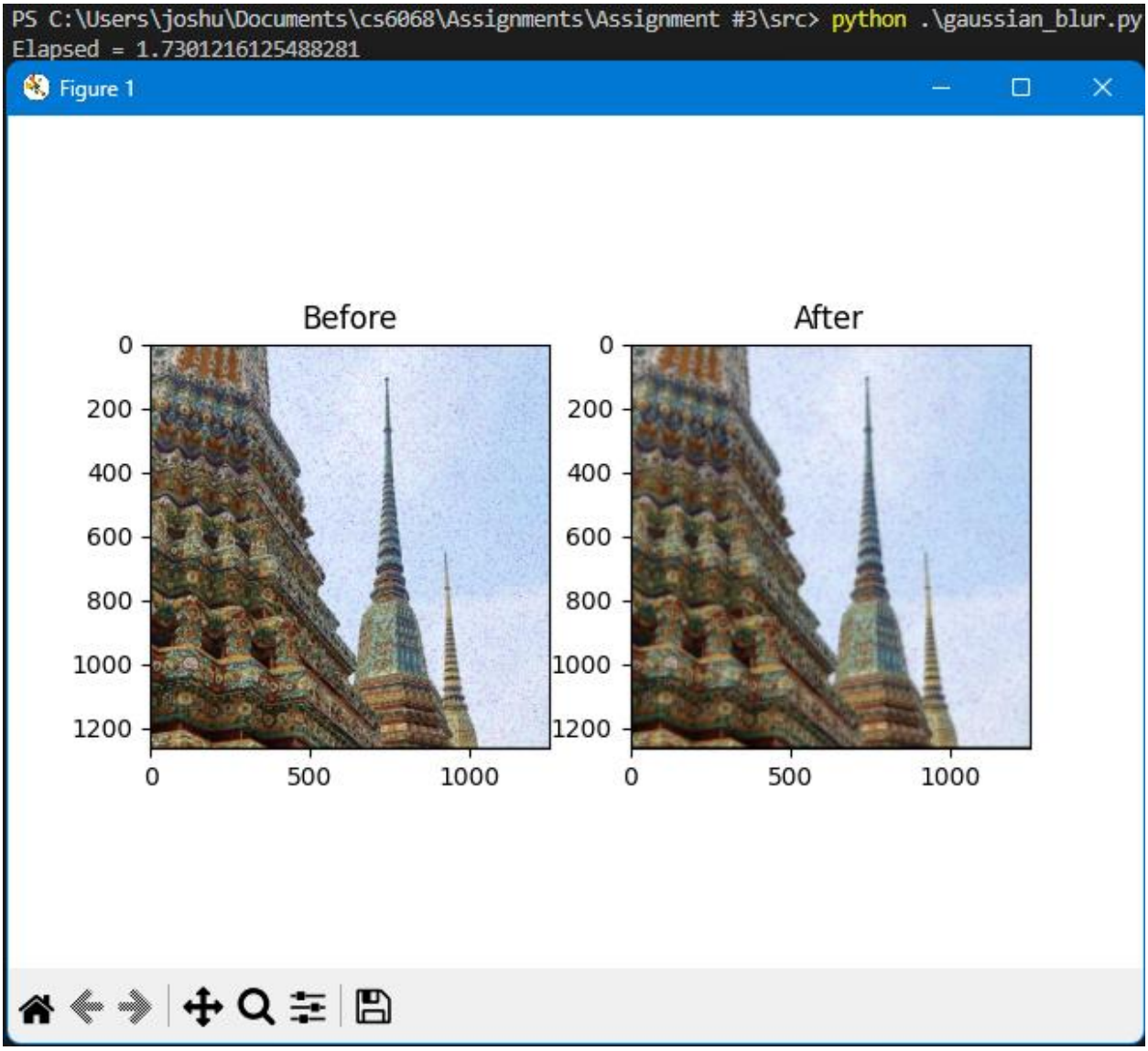

Figure 7: Gaussian Blur, Parallel, 9 Length

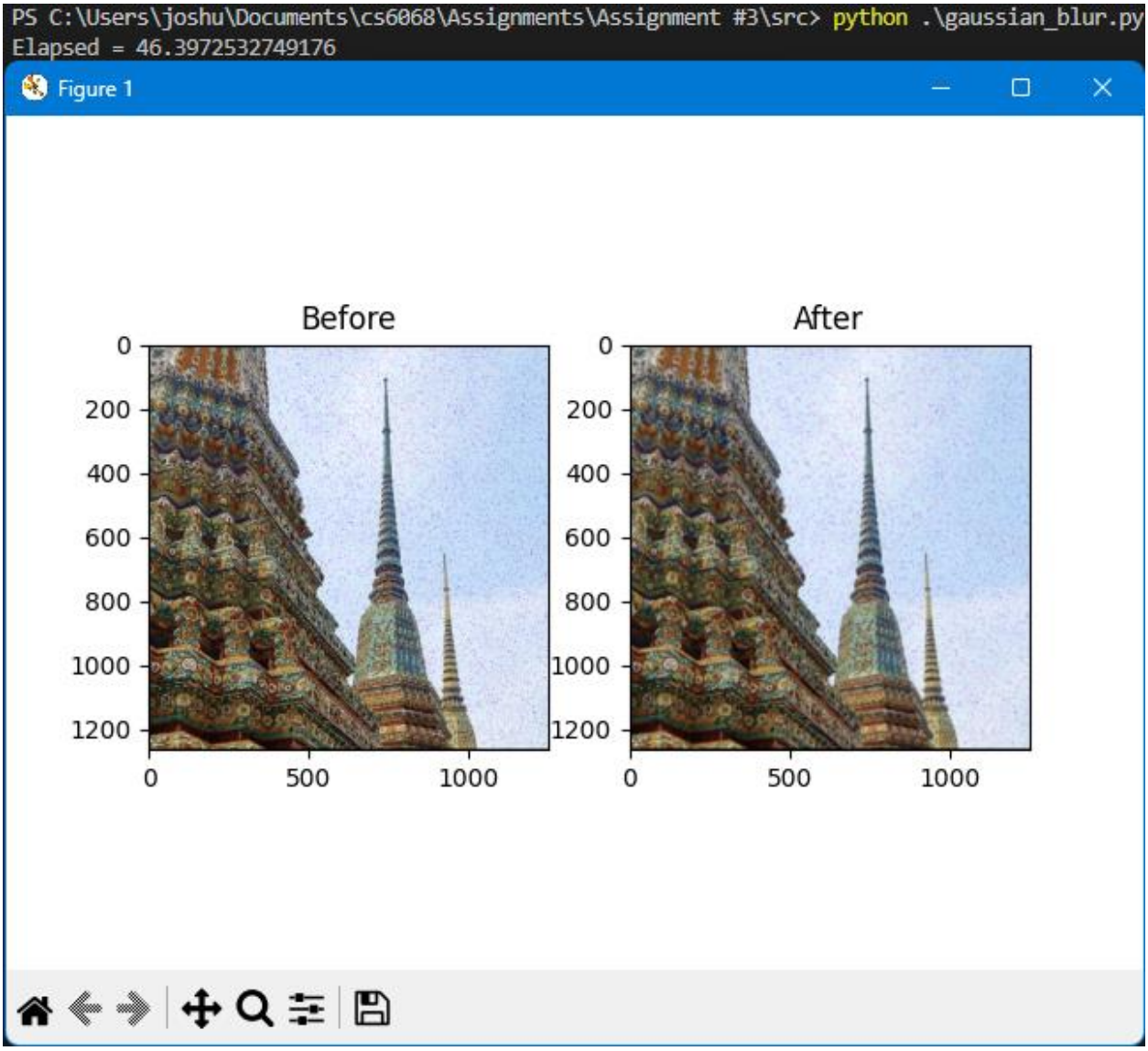

Figure 8: Gaussian Blur, Sequential, 5 Length

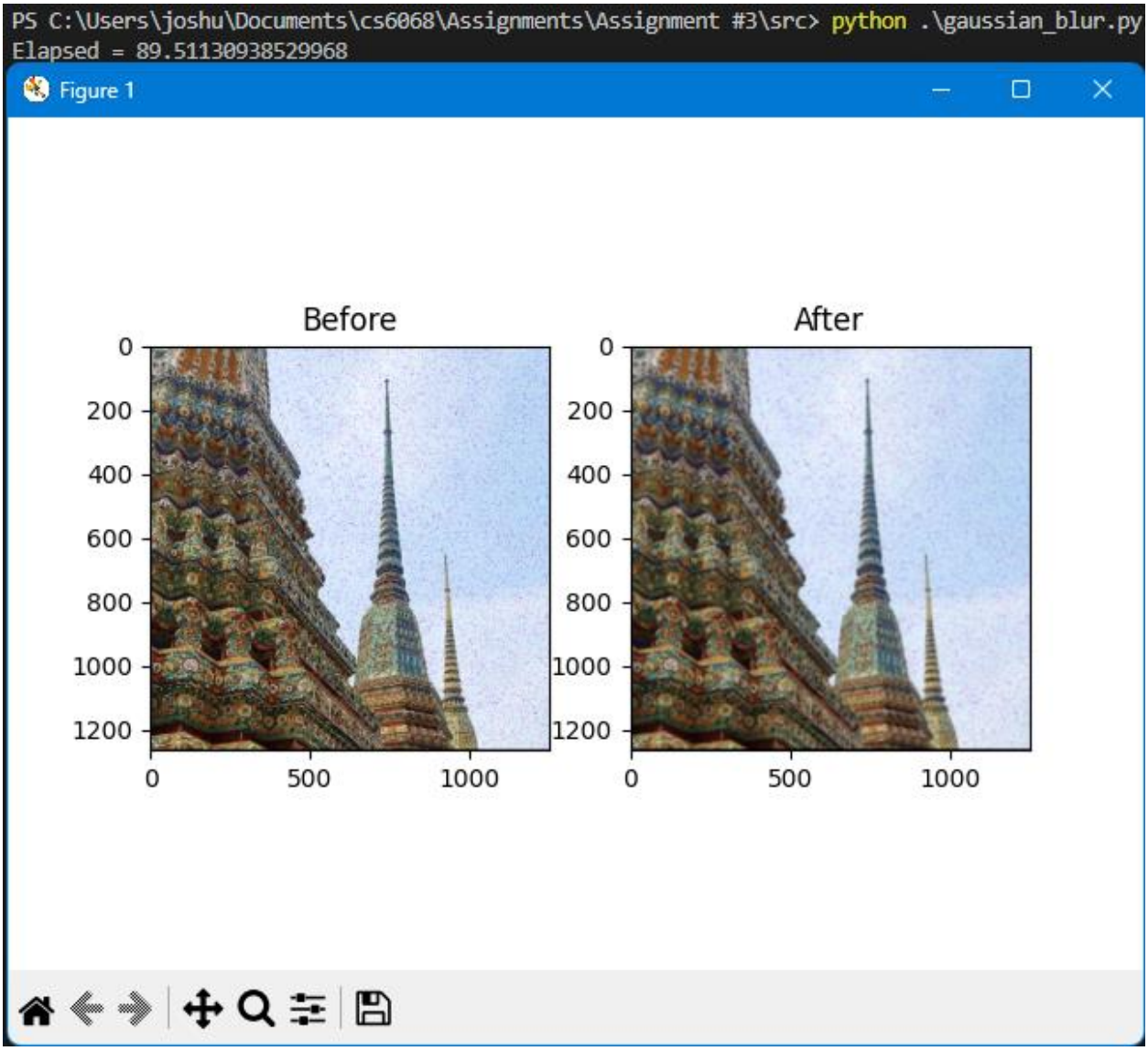

Figure 9: Gaussian Blur, Sequential, 7 Length

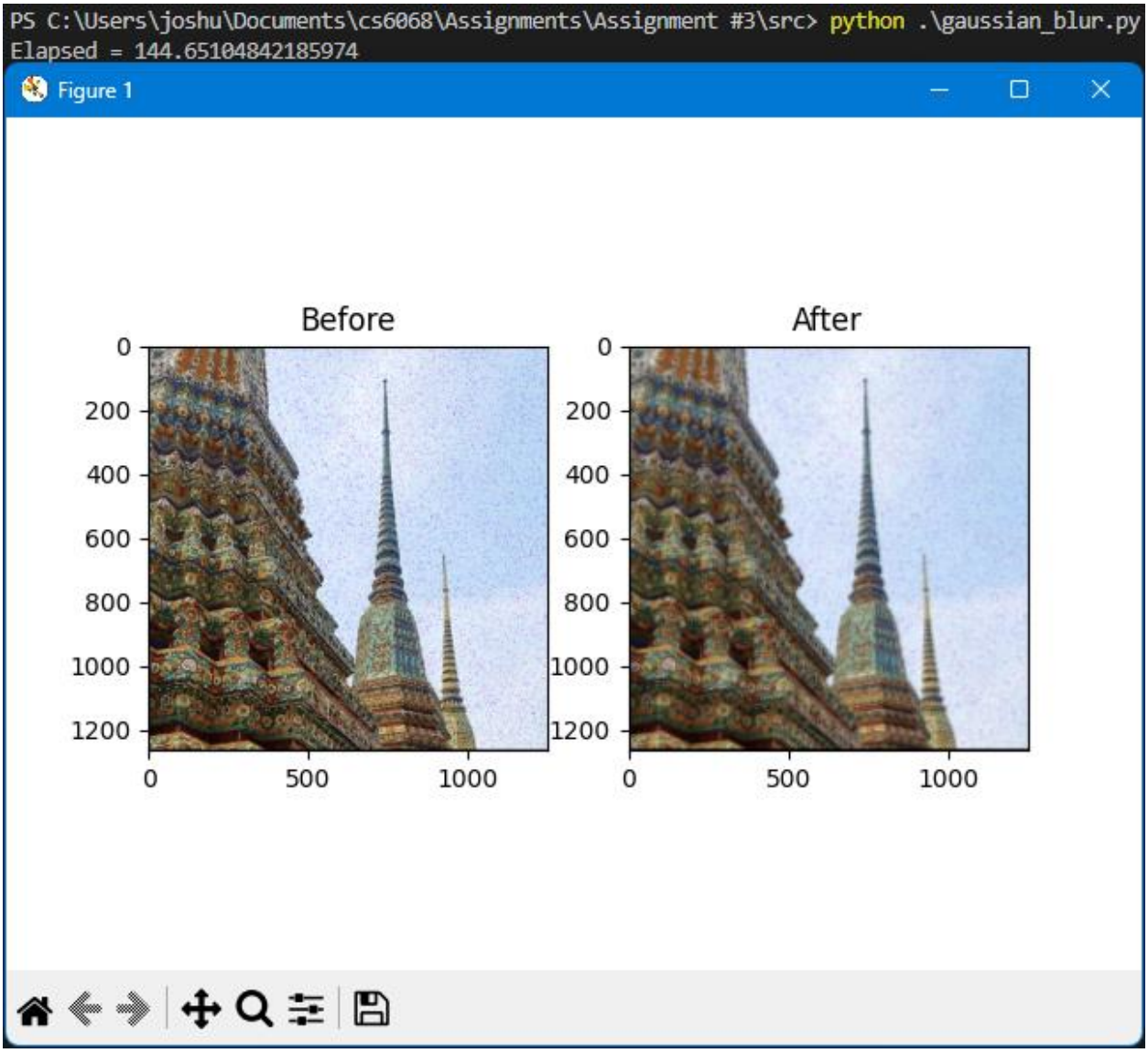

Figure 10: Gaussian Blur, Sequential, 9 Length

Conclusion

Numba was used to speed up execution of gaussian_blur.py, which blurred the noisy image using Gaussian blurring. Using the @jit decorator resulted in a speedup in all tests. In general, the speedup was larger for tests that performed more work (e.g., larger filter length).