

Lab Manual
of
High Performance Computing
(20DS509P)

By

Kavan Mistry
23MDS006



**DEPT. OF COMPUTER SCIENCE & ENGINEERING
SCHOOL OF TECHNOLOGY
PANDIT DEENDAYAL ENERGY UNIVERSITY
GANDHINAGAR, GUJARAT, INDIA
JANUARY-MAY, 2024**

Index

Sr. No .	Problem Statement	Date	Sign.
1.	<p>Write a program of matrix multiplication to demonstrate the performance enhancement done by parallelizing the code through Open MP threads.</p> <ul style="list-style-type: none"> Analyze the speedup and efficiency of the parallelized code. Vary the size of your matrices from 5, 50, 100, 500, 750, 1000, and 2000 and measure the runtime with one thread. For each matrix size, change the number of threads from 2, 4, 8, 10, 15, and 20 and plot the speedup versus the number of threads. Compute the efficiency. Display a visualization of performance comparison between serial, parallel and NumPY code. Explain whether or not the scaling behavior is as expected. 	17/1/2024	
2.	<p>Write a program for Leibniz series for PI calculation to demonstrate the performance enhancement done by parallelizing the code through Open MP work-sharing of loops. Display a visualization of performance comparison between serial and parallel, a visual analysis of delay/speedup with the help of varying thread counts and maximum terms in the series for Pi value calculation.</p> <ul style="list-style-type: none"> Implement the code with different thread count and different maximum number of terms to be calculated for the series such as thread count 10, 20 and terms 100, 1000, 10000, 1000000. Display a visualization of performance comparison between serial and parallel, a visual analysis of delay/speedup with the help of varying thread counts and maximum terms in the series for Pi value calculation. 	24/1/2024	
3.	Implement Producer-Consumer problem (PCP). Analyze the significance of semaphore, mutex, bounded buffer, producer thread, and consumer thread using the code available on Producer-Consumer Problem in Python - AskPython. Demonstrate how PCP occurs for an application of your choice.	29/1/2024	
4.	Write a program to generate and print Fibonacci series, one thread must generate the series up to number and other thread must print them. Ensure proper synchronization.	31/1/2024	
5.	Consider a scenario where a person visits a supermarket for shopping. S/He purchases various items in different sections such as clothing, grocery, utensils. Write an OpenMP program to process the bill parallelly	31/1/2024	

	in each section and display the final amount to be paid by the customer. Analyze the time taken by sequential and parallel processing.		
6.	<p>Implement the following programs of OpenMPI</p> <ul style="list-style-type: none"> • Print “Welcome to PDPU from process (processno_totalprocesses)”. • Apply denoising algorithm to a set of n images with 4 processes. (n=4, 8). • Analyze time taken by serial and openMPI processes. • Try for 100 or more number of images. 	7/2/2024	
7.	<ol style="list-style-type: none"> 1. Write a program to implement arithmetic calculations using MPI processes. 2. Write a program with different processes to apply following functions to an image in parallel. <ul style="list-style-type: none"> • Read an image. • Convert above RGB image to grayscale. • Find edges in the image. • Show the original image. 	14/2/2024	
8.	<p>Write a program to pass message from one process to another and print output.</p> <ul style="list-style-type: none"> • In synchronous communication • In asynchronous communication. Show using overlapping of task in non-blocking mode. 	14/2/2024	
9.	<p>Calculate Pi value using openMPI send and receive messages for atleast 35-40 terms.</p> <p>Try the below mentioned commands, explain their task in one line and paste the output for each of them</p> <ul style="list-style-type: none"> • Change the value of n as 2, 4, 8, 16. • Analyze the performance improvement using number of processes. 	14/2/2024	
10.	<p>Write a program to show collective communication by taking suitable example such that computing average of n numbers or computing sum or product of two matrices:</p> <ul style="list-style-type: none"> • Bcast function • Scatter function • Gather function 	14/2/2024	
11.	<ol style="list-style-type: none"> 1. Describe Canon's Matrix Multiplication algorithm. 2. Implement Canon's Matrix Multiplication using collective communication. 3. Analyze the efficiency of the code. 	19/2/2024	
12.	<ol style="list-style-type: none"> 1. Write about derived data types used in MPI programming. 2. Steps to create and use derived data types. 3. Write its uses. 	19/3/2024	

	4. Implement communication of derived data using one suitable example.			
13.	lshw (List Hardware) lsusb (List USB Devices) lspci (List PCI Devices) lsblk (List Block Devices) lscpu (List CPU) df (Disk Free) dmidecode (DMI Table Decode) ip a (IP Addresse)	top htop nvidia-smi lstopo perf numactl sar		
	For the given Python scripts that queries the CPU usage on a Linux-based system, understand the same and note the output for your device.		11/3/2024	
14.	Presentation of the respective topic assigned		26/3/2024	
15.	Perform the following Image Processing Operations using the given images: <ul style="list-style-type: none">• Image Blurring• Image Thresholding• Histogram based image analysis• Image Filtering/Denoising• Image Gray scaling		26/3/2024	
16.	Empirically understand and document the answers to the following: <ul style="list-style-type: none">• What is CUDA?• What is the prerequisite for learning CUDA?• What are the languages that support CUDA?• What do you mean by a CUDA ready architecture?• How CUDA works?• What are the benefits and limitations of CUDA programming?• Understand and explain the CUDA program structure with an example.• Explain CUDA thread organization• Install and try CUDA sample program and explain the same. (installation steps)		1/4/2024	
17.	Implement following CUDA programs: <ol style="list-style-type: none">1. To print hello message on the screen using kernal function2. To add two vectors of size 100 and 20000 and analyze the performance comparison between cpu and gpu processing3. To multiply two matrix of size 20 X 20 and 1024 X 1024 analyze the performance comparison between cpu and gpu processing4. To obtain CUDA device information and print the output		15/4/2024	
18.	Implement the following Image Processing operations in sequential and parallel using CUDA Programming. <ol style="list-style-type: none">1. Gaussian Blur		22/4/2024	

	<ul style="list-style-type: none"> • Describe Gaussian Blur in brief. • Where parallelism can be inserted? • Analyze the performance in serial and parallel model. <p>2. FFT- Fast Fourier Transform</p> <ul style="list-style-type: none"> • Describe FFT in brief. • Where parallelism can be inserted? • Analyze the performance in serial and parallel model. 		
19.	Final Learning Synopsis Submission		

Assignment 1

Write a program of matrix multiplication to demonstrate the performance enhancement done by parallelizing the code through Open MP threads. Analyze the speedup and efficiency of the parallelized code.

- Vary the size of your matrices from 5, 50, 100, 500, 750, 1000, and 2000 and measure the runtime with one thread.
- For each matrix size, change the number of threads from 2,4,8,10,15,20 and plot the speedup versus the number of threads. Compute the efficiency.
- Display a visualization of performance comparison between serial, parallel and NumPY code.
- Explain whether or not the scaling behavior is as expected.

```
[9]: import numpy as np
import threading
import time
import pandas as pd
import matplotlib.pyplot as plt
```

```
[10]: def multiply_matrix(A, B, result, start_row, end_row):
    try:
        for i in range(start_row, end_row):
            for j in range(N):
                result[i, j] = 0
                for k in range(N):
                    result[i, j] += A[i, k] * B[k, j]
    except NameError as e:
        pass
        # Handle the exception as per your requirements
    #     print(f"Exception in thread {threading.current_thread().name}: {e}")
```

```
[11]: def measure_time(matrix_size, num_threads=1):
    A = np.random.rand(matrix_size, matrix_size)
    B = np.random.rand(matrix_size, matrix_size)
    result = np.zeros((matrix_size, matrix_size))

    # Ensure at least one row per thread
```

```

chunk_size = max(1, matrix_size // num_threads)
threads = []

start_time = time.time()

for i in range(0, matrix_size, chunk_size):
    end_row = min(i + chunk_size, matrix_size)
    thread = threading.Thread(target=multiply_matrix, args=(A, B, result, i, end_row))
    thread.start()
    threads.append(thread)

for thread in threads:
    thread.join()

end_time = time.time()

return max(end_time - start_time, 1e-10)

```

```

[12]: def main():
    matrix_sizes = [5, 50, 100, 250, 500, 750, 1000, 2000, 5000]
    thread_counts = [1, 2, 4, 8, 10, 15, 20]

    results = []

    for size in matrix_sizes:
        serial_time = measure_time(size, num_threads=1)

        for threads in thread_counts:
            parallel_time = measure_time(size, num_threads=threads)
            speedup = serial_time / parallel_time
            efficiency = speedup / threads
            results.append({
                'Matrix Size': size,
                'Threads': threads,
                'Serial Time': serial_time,
                'Parallel Time': parallel_time,
                'Speedup': speedup,
                'Efficiency': efficiency
            })

    df = pd.DataFrame(results)
    df.to_csv('matrix_multiplication_results.csv', index=False)

```

```

[13]: if __name__ == "__main__":
    main()

```

```
[14]: df2 = pd.read_csv('matrix_multiplication_results.csv')
df2
```

```
[14]:   Matrix Size  Threads  Serial Time  Parallel Time    Speedup \
0            5         1  1.000000e-10  1.002073e-03  9.979310e-08
1            5         2  1.000000e-10  1.012325e-03  9.878248e-08
2            5         4  1.000000e-10  2.073288e-03  4.823257e-08
3            5         8  1.000000e-10  2.097368e-03  4.767880e-08
4            5        10  1.000000e-10  1.571417e-03  6.363684e-08
..          ...
58           5000       4  1.000000e-10  1.000000e-10  1.000000e+00
59           5000       8  1.000000e-10  9.087563e-03  1.100405e-08
60           5000      10  1.000000e-10  1.626253e-03  6.149104e-08
61           5000      15  1.000000e-10  9.224892e-03  1.084024e-08
62           5000      20  1.000000e-10  7.979870e-04  1.253153e-07

      Efficiency
0  9.979310e-08
1  4.939124e-08
2  1.205814e-08
3  5.959850e-09
4  6.363684e-09
..        ...
58  2.500000e-01
59  1.375506e-09
60  6.149104e-09
61  7.226824e-10
62  6.265766e-09

[63 rows x 6 columns]
```

```
[15]: def plot_matrix_size(df2, matrix_size):
    plt.figure(figsize=(15, 15))

    # Plot Serial Time
    plt.subplot(4, 1, 1)
    for size in matrix_sizes:
        data = df2[df2['Matrix Size'] == size]
        plt.plot(data['Threads'], data['Serial Time'], label=f'Matrix Size_{size}', marker='o')
    plt.title('Serial Time')
    plt.xlabel('Number of Threads')
    plt.ylabel('Serial Time (s)')
    plt.legend()

    # Plot Parallel Time
    plt.subplot(4, 1, 2)
```

```

for size in matrix_sizes:
    data = df2[df2['Matrix Size'] == size]
    plt.plot(data['Threads'], data['Parallel Time'], label=f'Matrix Size {size}', marker='o')
plt.title('Parallel Time')
plt.xlabel('Number of Threads')
plt.ylabel('Parallel Time (s)')
plt.legend()

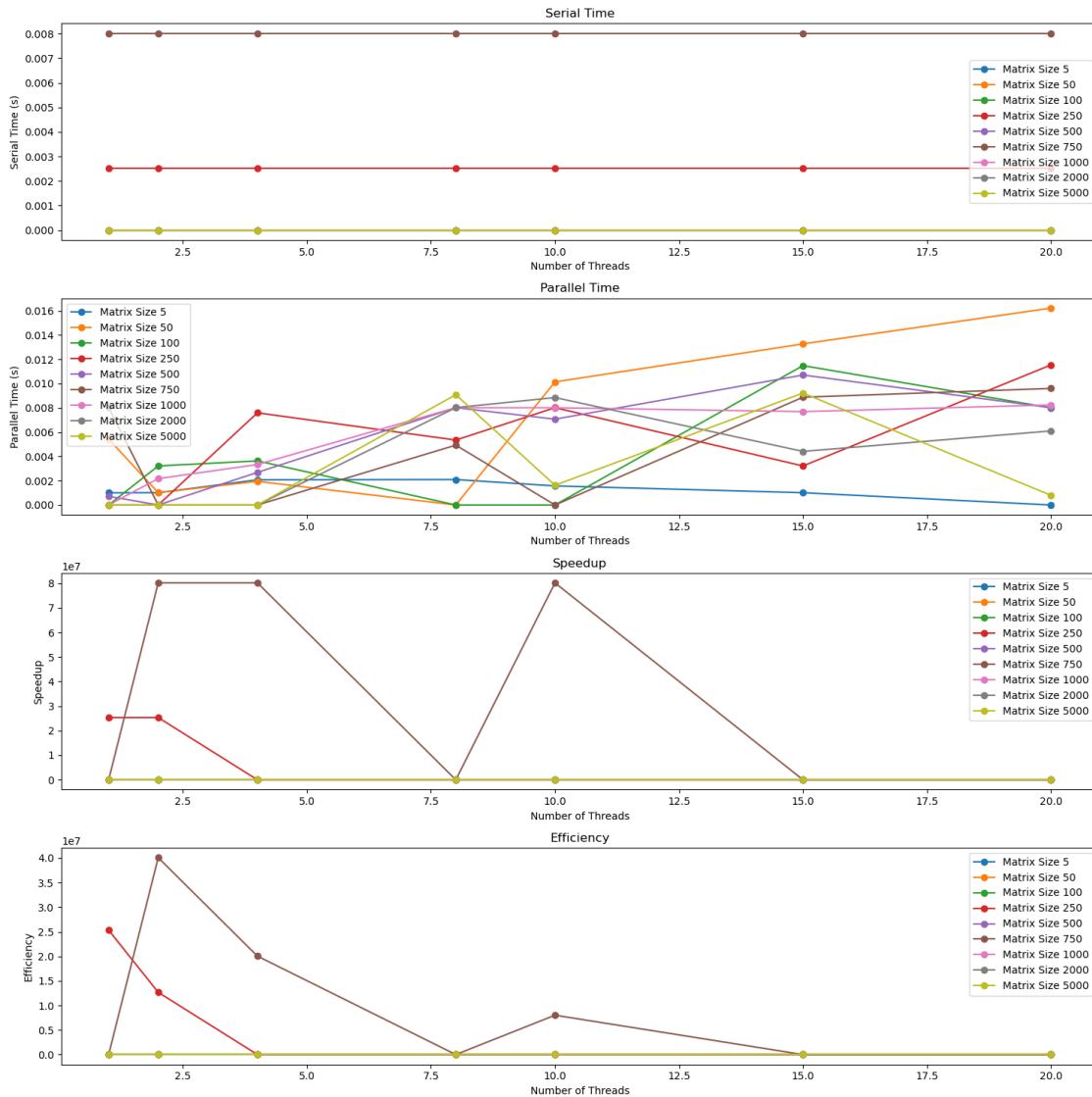
# Plot Speedup
plt.subplot(4, 1, 3)
for size in matrix_sizes:
    data = df2[df2['Matrix Size'] == size]
    plt.plot(data['Threads'], data['Speedup'], label=f'Matrix Size {size}', marker='o')
plt.title('Speedup')
plt.xlabel('Number of Threads')
plt.ylabel('Speedup')
plt.legend()

# Plot Efficiency
plt.subplot(4, 1, 4)
for size in matrix_sizes:
    data = df2[df2['Matrix Size'] == size]
    plt.plot(data['Threads'], data['Efficiency'], label=f'Matrix Size {size}', marker='o')
plt.title('Efficiency')
plt.xlabel('Number of Threads')
plt.ylabel('Efficiency')
plt.legend()

plt.tight_layout()
plt.show()

matrix_sizes = df2['Matrix Size'].unique()
plot_matrix_size(df2, matrix_sizes)

```



with using numpy

```
[16]: def measure_time_np(matrix_size):
    A = np.random.rand(matrix_size, matrix_size)
    B = np.random.rand(matrix_size, matrix_size)

    start_time = time.time()
    result = np.dot(A, B)
    end_time = time.time()

    return end_time - start_time
```

```
[17]: def main_np():
    matrix_sizes = [5, 50, 100, 250, 500, 750, 1000, 2000, 5000]
    thread_counts = [1, 2, 4, 8, 10, 15, 20]

    results_np = []

    for size in matrix_sizes:
        serial_time_np = measure_time_np(size)

        for threads in thread_counts:
            parallel_time_np = measure_time_np(size)

            # Ensure parallel_time_np is not zero before calculating speedup and efficiency
            if parallel_time_np != 0:
                speedup_np = serial_time_np / parallel_time_np
                efficiency_np = speedup_np / threads
            else:
                speedup_np = 0
                efficiency_np = 0

            results_np.append({
                'Matrix Size': size,
                'Threads': threads,
                'Serial Time': serial_time_np,
                'Parallel Time': parallel_time_np,
                'Speedup': speedup_np,
                'Efficiency': efficiency_np
            })

    df_np = pd.DataFrame(results_np)
    df_np.to_csv('matrix_multiplication_results_np2.csv', index=False)
```

```
[18]: if __name__ == "__main__":
    main_np()
```

```
[19]: df3 = pd.read_csv('matrix_multiplication_results_np2.csv')
df3
```

	Matrix Size	Threads	Serial Time	Parallel Time	Speedup	Efficiency
0	5	1	0.0000	0.000000	0.000000	0.000000
1	5	2	0.0000	0.000000	0.000000	0.000000
2	5	4	0.0000	0.000000	0.000000	0.000000
3	5	8	0.0000	0.000000	0.000000	0.000000
4	5	10	0.0000	0.000000	0.000000	0.000000
..
58	5000	4	3.1658	2.348449	1.348038	0.337010

59	5000	8	3.1658	1.465680	2.159953	0.269994
60	5000	10	3.1658	1.186051	2.669194	0.266919
61	5000	15	3.1658	1.195132	2.648913	0.176594
62	5000	20	3.1658	1.168196	2.709990	0.135500

[63 rows x 6 columns]

```
[20]: def plot_matrix_size(df3, matrix_size):
    plt.figure(figsize=(15, 15))

    # Plot Serial Time
    plt.subplot(4, 1, 1)
    for size in matrix_sizes:
        data = df3[df3['Matrix Size'] == size]
        plt.plot(data['Threads'], data['Serial Time'], label=f'Matrix Size {size}', marker='o')
    plt.title('Serial Time')
    plt.xlabel('Number of Threads')
    plt.ylabel('Serial Time (s)')
    plt.legend()

    # Plot Parallel Time
    plt.subplot(4, 1, 2)
    for size in matrix_sizes:
        data = df3[df3['Matrix Size'] == size]
        plt.plot(data['Threads'], data['Parallel Time'], label=f'Matrix Size {size}', marker='o')
    plt.title('Parallel Time')
    plt.xlabel('Number of Threads')
    plt.ylabel('Parallel Time (s)')
    plt.legend()

    # Plot Speedup
    plt.subplot(4, 1, 3)
    for size in matrix_sizes:
        data = df3[df3['Matrix Size'] == size]
        plt.plot(data['Threads'], data['Speedup'], label=f'Matrix Size {size}', marker='o')
    plt.title('Speedup')
    plt.xlabel('Number of Threads')
    plt.ylabel('Speedup')
    plt.legend()

    # Plot Efficiency
    plt.subplot(4, 1, 4)
    for size in matrix_sizes:
        data = df3[df3['Matrix Size'] == size]
```

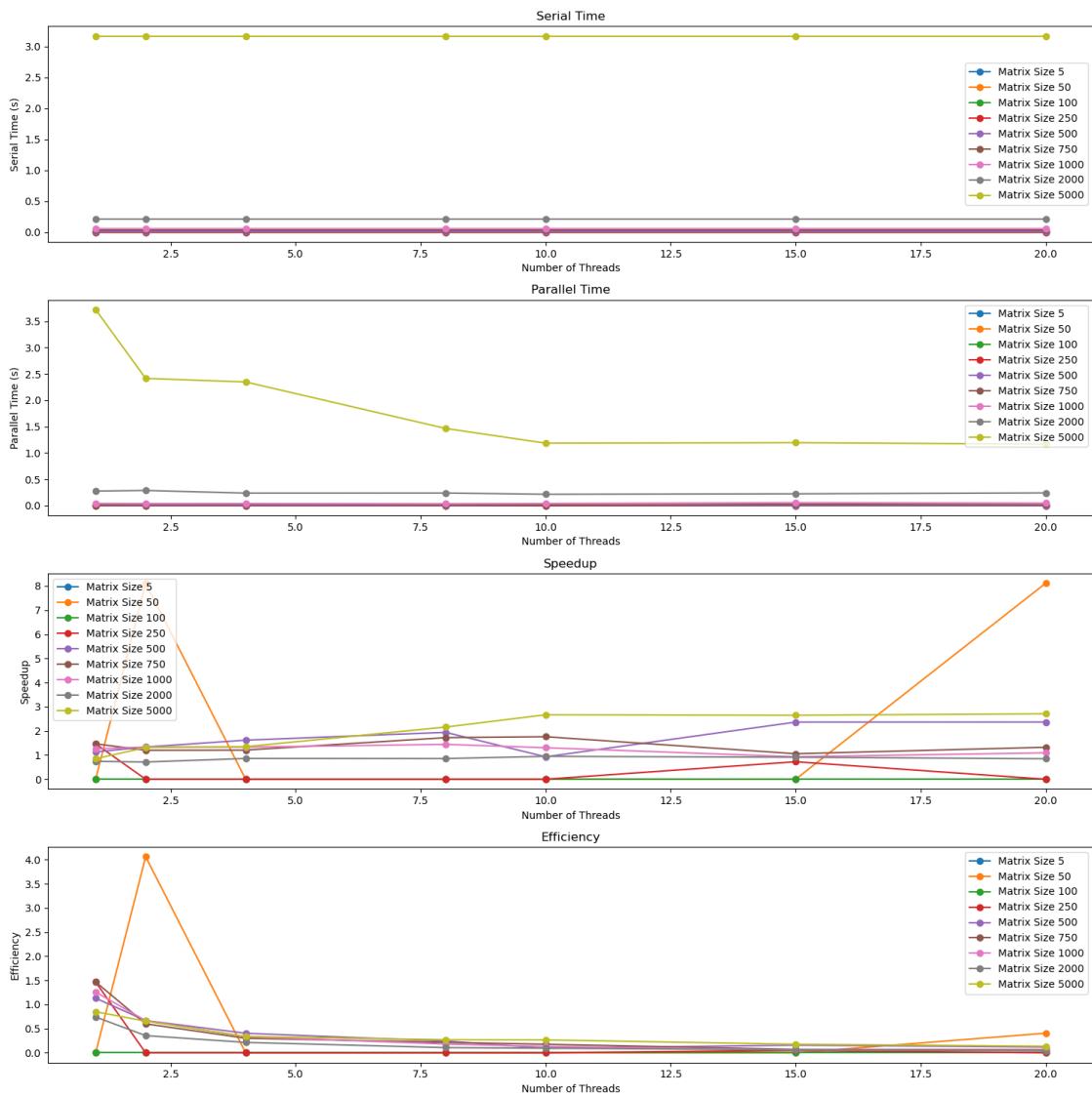
```

plt.plot(data['Threads'], data['Efficiency'], label=f'Matrix Size {size}', marker='o')
plt.title('Efficiency')
plt.xlabel('Number of Threads')
plt.ylabel('Efficiency')
plt.legend()

plt.tight_layout()
plt.show()

matrix_sizes = df3['Matrix Size'].unique()
plot_matrix_size(df3, matrix_sizes)

```



Q. Explain whether or not the scaling behavior is as expected.

- Increasing the number of threads in the manual loop implementation results in decreasing gains in speed and efficiency, particularly for bigger matrix sizes. The numPY implementation, on the other hand, shows better scaling behavior, with notable increases in speed and efficiency as the number of threads rises, suggesting a more successful parallelization approach. so we can say the scaling behavior is as expected.

Assignment 2

Write a program for Leibniz series for PI calculation to demonstrate the performance enhancement done by parallelizing the code through Open MP work-sharing of loops.

- Implement the code with different thread count and different maximum number of terms to be calculated for the series such as thread count 10, 20 and terms 100, 1000, 10000, 1000000.
- Display a visualization of performance comparison between serial and parallel, a visual analysis of delay/speedup with the help of varying thread counts and maximum terms in the series for Pi value calculation.

```
[1]: import numpy as np
import threading
import time
import pandas as pd
import matplotlib.pyplot as plt
```

```
[2]: # sequential using leibniz method
```

```
[3]: def estimate_pi_leibniz_sequential(num_terms):
    pi_estimate = 0

    start_time = time.time()

    for k in range(num_terms):
        term = (-1) ** k / (2 * k + 1)
        pi_estimate += term

    pi_estimate *= 4

    end_time = time.time()
    elapsed_time = end_time - start_time

    return pi_estimate, elapsed_time
```

```
[4]: num_terms_list = [100, 500, 700, 1000, 5000, 10000, 1000000]
pi_estimates = []
elapsed_times = []
```

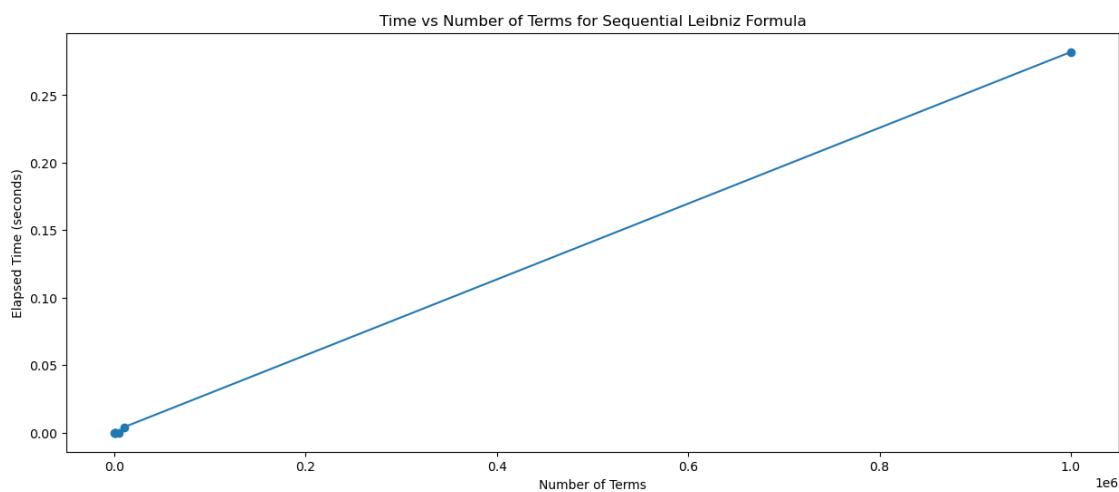
```
[5]: for num_terms in num_terms_list:
    pi_estimate, elapsed_time = estimate_pi_leibniz_sequential(num_terms)
    pi_estimates.append(pi_estimate)
    elapsed_times.append(elapsed_time)
    print(f"Number of Terms: {num_terms}, Estimated Pi: {pi_estimate}, Elapsed Time: {elapsed_time:.6f} seconds")
```

Number of Terms: 100, Estimated Pi: 3.1315929035585537, Elapsed Time: 0.000000 seconds
 Number of Terms: 500, Estimated Pi: 3.139592655589785, Elapsed Time: 0.000000 seconds
 Number of Terms: 700, Estimated Pi: 3.1401640828900845, Elapsed Time: 0.000000 seconds
 Number of Terms: 1000, Estimated Pi: 3.140592653839794, Elapsed Time: 0.000000 seconds
 Number of Terms: 5000, Estimated Pi: 3.141392653591791, Elapsed Time: 0.000000 seconds
 Number of Terms: 10000, Estimated Pi: 3.1414926535900345, Elapsed Time: 0.004056 seconds
 Number of Terms: 1000000, Estimated Pi: 3.1415916535897743, Elapsed Time: 0.282051 seconds

```
[6]: plt.figure(figsize=(15, 6))

plt.plot(num_terms_list, elapsed_times, marker='o')
plt.title('Time vs Number of Terms for Sequential Leibniz Formula')
plt.xlabel('Number of Terms')
plt.ylabel('Elapsed Time (seconds)')

plt.show()
```



```
[7]: # sequential using math module
```

```
[8]: import math
```

```
[9]: def calculate_pi():
    start_time = time.time()

    # Accessing the built-in pi constant
    pi_estimate = math.pi

    end_time = time.time()
    elapsed_time = end_time - start_time

    return pi_estimate, elapsed_time
```

```
[10]: num_terms_list = [100, 500, 1000, 5000, 10000, 1000000]
pi_estimates_builtin = []
elapsed_times_builtin = []
```

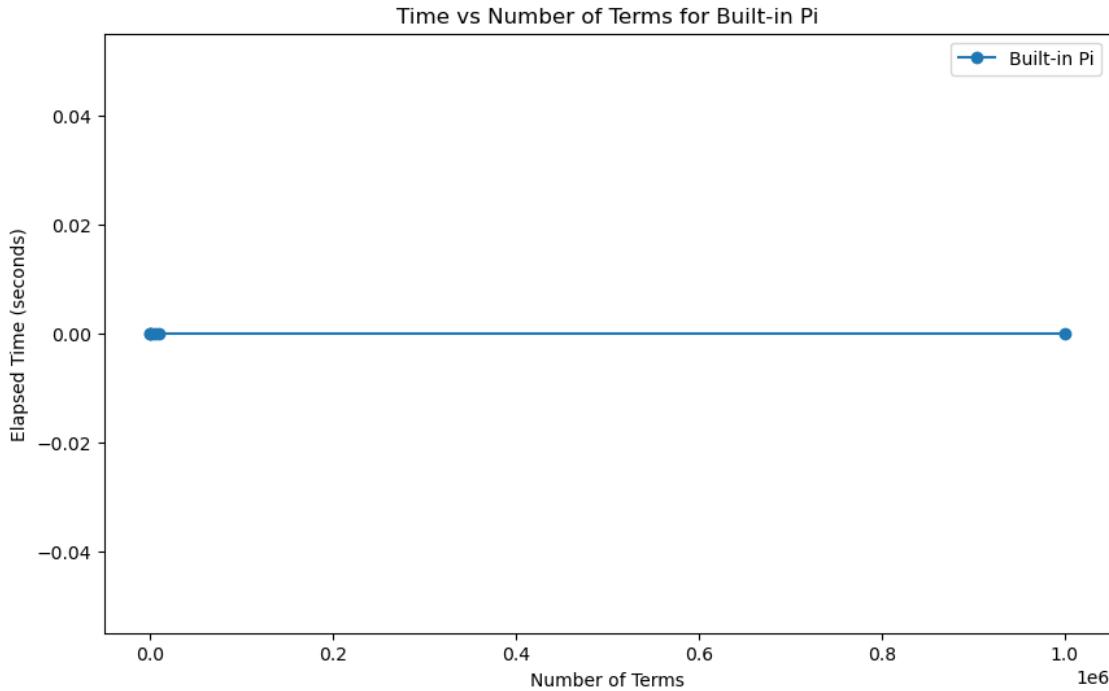
```
[11]: for num_terms in num_terms_list:
    pi_estimate, elapsed_time = calculate_pi()
    pi_estimates_builtin.append(pi_estimate)
    elapsed_times_builtin.append(elapsed_time)
    print(f"Number of Terms: {num_terms}, Built-in Pi: {pi_estimate}, Elapsed Time: {elapsed_time:.6f} seconds")
```

Number of Terms: 100, Built-in Pi: 3.141592653589793, Elapsed Time: 0.000000 seconds
Number of Terms: 500, Built-in Pi: 3.141592653589793, Elapsed Time: 0.000000 seconds
Number of Terms: 1000, Built-in Pi: 3.141592653589793, Elapsed Time: 0.000000 seconds
Number of Terms: 5000, Built-in Pi: 3.141592653589793, Elapsed Time: 0.000000 seconds
Number of Terms: 10000, Built-in Pi: 3.141592653589793, Elapsed Time: 0.000000 seconds
Number of Terms: 1000000, Built-in Pi: 3.141592653589793, Elapsed Time: 0.000000 seconds

```
[12]: # Plotting the results
plt.figure(figsize=(10, 6))

plt.plot(num_terms_list, elapsed_times_builtin, marker='o', label='Built-in Pi')
plt.title('Time vs Number of Terms for Built-in Pi')
plt.xlabel('Number of Terms')
plt.ylabel('Elapsed Time (seconds)')
plt.legend()
```

```
plt.show()
```



```
[13]: # parallel processing using leibniz method
```

```
[14]: def estimate_pi_leibniz_parallel(num_terms, num_threads):
    pi_estimate = 0
    terms_per_thread = num_terms // num_threads
    threads = []

    def calculate_terms(start, end):
        nonlocal pi_estimate
        for k in range(start, end):
            term = (-1) ** k / (2 * k + 1)
            pi_estimate += term

    start_time = time.time()

    for i in range(num_threads):
        start = i * terms_per_thread
        end = (i + 1) * terms_per_thread if i < num_threads - 1 else num_terms
        thread = threading.Thread(target=calculate_terms, args=(start, end))
        thread.start()
        threads.append(thread)
```

```

    for thread in threads:
        thread.join()

    pi_estimate *= 4

    end_time = time.time()
    elapsed_time = end_time - start_time

    return pi_estimate, elapsed_time

```

[15]:

```

num_terms_list = [100, 500, 1000, 5000, 10000, 1000000]
num_threads_list = [2, 4, 8, 10, 20]
pi_estimates_parallel = []
elapsed_times_parallel = []

```

[16]:

```

for num_terms in num_terms_list:
    for num_threads in num_threads_list:
        pi_estimate, elapsed_time = estimate_pi_leibniz_parallel(num_terms, num_threads)
        pi_estimates_parallel.append(pi_estimate)
        elapsed_times_parallel.append(elapsed_time)
        print(f"Number of Terms: {num_terms}, Threads: {num_threads}, Estimated Pi: {pi_estimate}, Elapsed Time: {elapsed_time:.6f} seconds")

```

Number of Terms: 100, Threads: 2, Estimated Pi: 3.1315929035585537, Elapsed Time: 0.004011 seconds
Number of Terms: 100, Threads: 4, Estimated Pi: 3.1315929035585537, Elapsed Time: 0.000000 seconds
Number of Terms: 100, Threads: 8, Estimated Pi: 3.1315929035585537, Elapsed Time: 0.000000 seconds
Number of Terms: 100, Threads: 10, Estimated Pi: 3.1315929035585537, Elapsed Time: 0.005803 seconds
Number of Terms: 100, Threads: 20, Estimated Pi: 3.1315929035585537, Elapsed Time: 0.002507 seconds
Number of Terms: 500, Threads: 2, Estimated Pi: 3.139592655589785, Elapsed Time: 0.004007 seconds
Number of Terms: 500, Threads: 4, Estimated Pi: 3.139592655589785, Elapsed Time: 0.000000 seconds
Number of Terms: 500, Threads: 8, Estimated Pi: 3.139592655589785, Elapsed Time: 0.000000 seconds
Number of Terms: 500, Threads: 10, Estimated Pi: 3.139592655589785, Elapsed Time: 0.000000 seconds
Number of Terms: 500, Threads: 20, Estimated Pi: 3.139592655589785, Elapsed Time: 0.004000 seconds
Number of Terms: 1000, Threads: 2, Estimated Pi: 3.140592653839794, Elapsed Time: 0.000000 seconds
Number of Terms: 1000, Threads: 4, Estimated Pi: 3.140592653839794, Elapsed Time: 0.000000 seconds

```

Number of Terms: 1000, Threads: 8, Estimated Pi: 3.140592653839794, Elapsed
Time: 0.000000 seconds
Number of Terms: 1000, Threads: 10, Estimated Pi: 3.140592653839794, Elapsed
Time: 0.005448 seconds
Number of Terms: 1000, Threads: 20, Estimated Pi: 3.140592653839794, Elapsed
Time: 0.003509 seconds
Number of Terms: 5000, Threads: 2, Estimated Pi: 3.141392653591791, Elapsed
Time: 0.000000 seconds
Number of Terms: 5000, Threads: 4, Estimated Pi: 3.141392653591791, Elapsed
Time: 0.000000 seconds
Number of Terms: 5000, Threads: 8, Estimated Pi: 3.141392653591791, Elapsed
Time: 0.004584 seconds
Number of Terms: 5000, Threads: 10, Estimated Pi: 3.141392653591791, Elapsed
Time: 0.000000 seconds
Number of Terms: 5000, Threads: 20, Estimated Pi: 3.141392653591791, Elapsed
Time: 0.003506 seconds
Number of Terms: 10000, Threads: 2, Estimated Pi: 3.1414926535900345, Elapsed
Time: 0.005043 seconds
Number of Terms: 10000, Threads: 4, Estimated Pi: 3.1414926535900345, Elapsed
Time: 0.003009 seconds
Number of Terms: 10000, Threads: 8, Estimated Pi: 3.1414926535900345, Elapsed
Time: 0.000000 seconds
Number of Terms: 10000, Threads: 10, Estimated Pi: 3.1414926535900345, Elapsed
Time: 0.008009 seconds
Number of Terms: 10000, Threads: 20, Estimated Pi: 3.1414926535900345, Elapsed
Time: 0.005652 seconds
Number of Terms: 1000000, Threads: 2, Estimated Pi: 3.1415916535897734, Elapsed
Time: 0.324527 seconds
Number of Terms: 1000000, Threads: 4, Estimated Pi: 3.1415916535897743, Elapsed
Time: 0.308578 seconds
Number of Terms: 1000000, Threads: 8, Estimated Pi: 3.1415916535897734, Elapsed
Time: 0.317601 seconds
Number of Terms: 1000000, Threads: 10, Estimated Pi: 3.141591653589775, Elapsed
Time: 0.291449 seconds
Number of Terms: 1000000, Threads: 20, Estimated Pi: 3.1415916535897743, Elapsed
Time: 0.299970 seconds

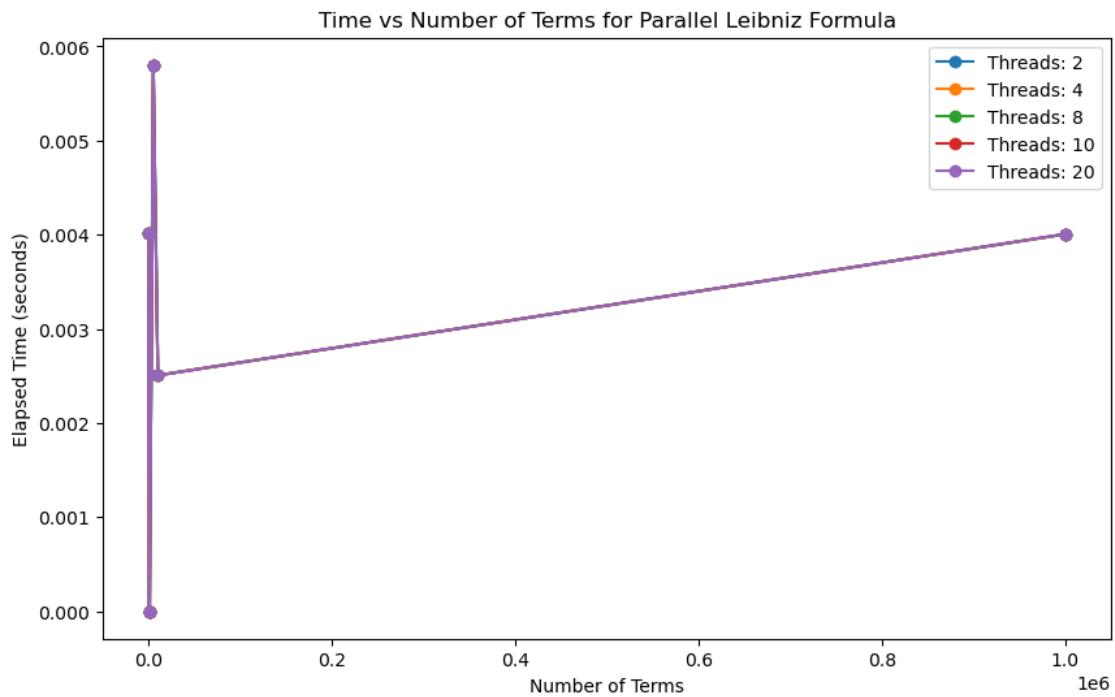
```

```
[17]: plt.figure(figsize=(10, 6))

for num_threads in num_threads_list:
    plt.plot(num_terms_list, elapsed_times_parallel[:len(num_terms_list)], 
             marker='o', label=f'Threads: {num_threads}')

plt.title('Time vs Number of Terms for Parallel Leibniz Formula')
plt.xlabel('Number of Terms')
plt.ylabel('Elapsed Time (seconds)')
plt.legend()
```

```
plt.show()
```



Assignment 3

1. Implement Producer-Consumer problem (PCP). Analyze the significance of semaphore, mutex, bounded buffer, producer thread, consumer thread using the code available on Producer-Consumer Problem in Python - AskPython.
 - (a) Write a brief about the problem and solution.
 - (b) Code and Output
2. Demonstrate how PCP occurs for a application of your choice.

Ans.

The producer-consumer problem is a classic synchronization problem in OS, especially in concurrent programming and multi-threading. It involves two types of processes:

1. Producers: These processes generate data or items and store them in the shared buffer;
2. Consumers: These processes retrieve and consume items from the buffer.

The challenge is to make sure that the following happen:
* Producers do not produce items if the buffer is full
* Consumers do not consume items if the buffer is empty.

The main objective is to keep producers and consumers in sync so as to avoid problems like data corruption, race conditions, and deadlocks.

Solution:

Here is a straightforward solution that makes use of bounded buffers and semaphores:

1. Shared Buffer:
 - A fixed-size buffer is shared between producers and consumers.
2. Semaphore for Empty Slots (empty):
 - Initialized to the size of the buffer.
 - Represents the number of empty slots in the buffer.
 - Decrement by producers when they add an item.
 - Decrement by consumers when they remove an item.
3. Semaphore for Full Slots (full):
 - Initialized to 0.
 - Represents the number of filled slots in the buffer.
 - Incremented by producers when they add an item.
 - Decrement by consumers when they remove an item.

4. Mutex Lock:

- Protects access to the shared buffer to race conditions.

```
[2]: import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```
[3]: import threading
import time
```

```
[4]: # Shared Memory variables
CAPACITY = 10
buffer = [-1 for i in range(CAPACITY)]
in_index = 0
out_index = 0
```

```
[5]: # Declaring Semaphores
mutex = threading.Semaphore()
empty = threading.Semaphore(CAPACITY)
full = threading.Semaphore(0)
```

```
[6]: # Producer Thread Class
class Producer(threading.Thread):
    def run(self):

        global CAPACITY, buffer, in_index, out_index
        global mutex, empty, full

        items_produced = 0
        counter = 0

        while items_produced < 20:
            empty.acquire()
            mutex.acquire()

            counter += 1
            buffer[in_index] = counter
            in_index = (in_index + 1)%CAPACITY
            print("Producer produced : ", counter)

            mutex.release()
            full.release()

            time.sleep(0)

        items_produced += 1
```

```
[7]: # Consumer Thread Class
class Consumer(threading.Thread):
    def run(self):

        global CAPACITY, buffer, in_index, out_index, counter
        global mutex, empty, full

        items_consumed = 0

        while items_consumed < 20:
            full.acquire()
            mutex.acquire()

            item = buffer[out_index]
            out_index = (out_index + 1)%CAPACITY
            print("Consumer consumed item : ", item)

            mutex.release()
            empty.release()

            time.sleep(0.5)

        items_consumed += 1
```

```
[8]: producer = Producer()
consumer = Consumer()

consumer.start()
producer.start()

producer.join()
consumer.join()
```

```
Producer produced : 1
Consumer consumed item : 1
Producer produced : 2
Producer produced : 3
Producer produced : 4
Producer produced : 5
Producer produced : 6
Producer produced : 7
Producer produced : 8
Producer produced : 9
Producer produced : 10
Producer produced : 11
Consumer consumed item : 2
Producer produced : 12
Consumer consumed item : 3
```

```
Producer produced : 13
Consumer consumed item : 4
Producer produced : 14
Consumer consumed item : 5
Producer produced : 15
Consumer consumed item : 6
Producer produced : 16
Consumer consumed item : 7
Producer produced : 17
Consumer consumed item : 8
Producer produced : 18
Consumer consumed item : 9
Producer produced : 19
Consumer consumed item : 10
Producer produced : 20
Consumer consumed item : 11
Consumer consumed item : 12
Consumer consumed item : 13
Consumer consumed item : 14
Consumer consumed item : 15
Consumer consumed item : 16
Consumer consumed item : 17
Consumer consumed item : 18
Consumer consumed item : 19
Consumer consumed item : 20
```

Implementation use if else and for loop

```
[9]: import time
```

```
[10]: CAPACITY = 10
buffer = [-1 for i in range(CAPACITY)]
in_index = 0
out_index = 0
```

```
[11]: mutex = threading.Semaphore()
empty = threading.Semaphore(CAPACITY)
full = threading.Semaphore(0)
```

```
[12]: class Producer(threading.Thread):
    def run(self):
        global CAPACITY, buffer, in_index
        global mutex, empty, full

        for counter in range(1, 21):
            empty.acquire()
            mutex.acquire()
```

```
    buffer[in_index] = counter
    in_index = (in_index + 1) % CAPACITY
    print("Producer produced:", counter)

    mutex.release()
    full.release()

    time.sleep(0)
```

```
[13]: class Consumer(threading.Thread):
    def run(self):
        global CAPACITY, buffer, out_index
        global mutex, empty, full

        for _ in range(20):
            full.acquire()
            mutex.acquire()

            item = buffer[out_index]
            out_index = (out_index + 1) % CAPACITY
            print("Consumer consumed item:", item)

            mutex.release()
            empty.release()

            time.sleep(0.5)
```

```
[14]: producer = Producer()
consumer = Consumer()

consumer.start()
producer.start()

producer.join()
consumer.join()
```

```
Producer produced: 1
Consumer consumed item: 1
Producer produced: 2
Producer produced: 3
Producer produced: 4
Producer produced: 5
Producer produced: 6
Producer produced: 7
Producer produced: 8
Producer produced: 9
Producer produced: 10
Producer produced: 11
```

```
Consumer consumed item: 2
Producer produced: 12
Consumer consumed item: 3
Producer produced: 13
Consumer consumed item: 4
Producer produced: 14
Consumer consumed item: 5
Producer produced: 15
Consumer consumed item: 6
Producer produced: 16
Consumer consumed item: 7
Producer produced: 17
Consumer consumed item: 8
Producer produced: 18
Consumer consumed item: 9
Producer produced: 19
Consumer consumed item: 10
Producer produced: 20
Consumer consumed item: 11
Consumer consumed item: 12
Consumer consumed item: 13
Consumer consumed item: 14
Consumer consumed item: 15
Consumer consumed item: 16
Consumer consumed item: 17
Consumer consumed item: 18
Consumer consumed item: 19
Consumer consumed item: 20
```

example usage

```
[15]: import queue
      import random
```

Example

Event Handling in GUI:

- Producers: User input events (clicks, keystrokes).
- Consumers: Event handlers or listeners.
- Buffer: Event queue.

```
[16]: MAX_QUEUE_SIZE = 5
event_queue = queue.Queue(MAX_QUEUE_SIZE)
mutex = threading.Lock()
empty = threading.Semaphore(MAX_QUEUE_SIZE)
full = threading.Semaphore(0)
```

```
[17]: class UserClickProducer(threading.Thread):
    def run(self):
        global MAX_QUEUE_SIZE, event_queue
        global mutex, empty, full

        for _ in range(10):
            print("User clicked")

            empty.acquire()
            mutex.acquire()

            event_queue.put("Click")

            mutex.release()
            full.release()

            time.sleep(random.uniform(0.1, 0.5))
```

```
[18]: class EventHandlerConsumer(threading.Thread):
    def run(self):
        global MAX_QUEUE_SIZE, event_queue
        global mutex, empty, full

        for _ in range(10):
            full.acquire()
            mutex.acquire()

            event = event_queue.get()
            print(f"Handling event: {event}")

            mutex.release()
            empty.release()

            time.sleep(random.uniform(0.1, 0.5))
```

```
[19]: user_click_producer = UserClickProducer()
event_handler_consumer = EventHandlerConsumer()

user_click_producer.start()
event_handler_consumer.start()

user_click_producer.join()
event_handler_consumer.join()
```

```
User clicked
Handling event: Click
User clicked
Handling event: Click
```

```
User clicked
Handling event: Click
User clicked
Handling event: Click
User clicked
User clicked
User clicked
User clicked
Handling event: Click
Handling event: Click
User clicked
Handling event: Click
User clicked
Handling event: Click
Handling event: Click
Handling event: Click
```

Assignment 4

Write a program to generate and print Fibonacci series, one thread must generate the series upto number and other thread must print them. Ensure proper synchronization.

```
[1]: import threading  
import time
```

```
[2]: def generate_fibonacci(n, fib_list, lock, start_time):  
    a, b = 0, 1  
    for _ in range(n):  
        with lock:  
            fib_list.append((a, time.time() - start_time))  
            a, b = b, a + b
```

```
[3]: def print_fibonacci(fib_list, lock):  
    with lock:  
        for entry in fib_list:  
            thread_id = threading.current_thread().ident  
            current_time = time.strftime("%Y-%m-%d %H:%M:%S", time.localtime())  
            fibonacci_value, time_taken = entry  
            print(f"fibonacci_value}, Thread ID {thread_id} at {current_time}:  
            , Time Taken: {time_taken:.6f} seconds")
```

```
[4]: def main():  
    n = int(input("Enter the number of Fibonacci numbers to generate: "))  
  
    fib_list = []  
    lock = threading.Lock()  
    start_time = time.time()  
  
    # Create two threads  
    generate_thread = threading.Thread(target=generate_fibonacci, args=(n,  
    ↪fib_list, lock, start_time))  
    print_thread = threading.Thread(target=print_fibonacci, args=(fib_list,  
    ↪lock))  
  
    # Start the threads
```

```

generate_thread.start()
print_thread.start()

# Wait for both threads to finish
generate_thread.join()
print_thread.join()

if __name__ == "__main__":
    main()

```

Enter the number of Fibonacci numbers to generate: 15

```

0, Thread ID 21104 at 2024-01-31 22:05:23: , Time Taken: 0.001002 seconds
1, Thread ID 21104 at 2024-01-31 22:05:23: , Time Taken: 0.001002 seconds
1, Thread ID 21104 at 2024-01-31 22:05:23: , Time Taken: 0.001002 seconds
2, Thread ID 21104 at 2024-01-31 22:05:23: , Time Taken: 0.001002 seconds
3, Thread ID 21104 at 2024-01-31 22:05:23: , Time Taken: 0.001002 seconds
5, Thread ID 21104 at 2024-01-31 22:05:23: , Time Taken: 0.001002 seconds
8, Thread ID 21104 at 2024-01-31 22:05:23: , Time Taken: 0.001002 seconds
13, Thread ID 21104 at 2024-01-31 22:05:23: , Time Taken: 0.001002 seconds
21, Thread ID 21104 at 2024-01-31 22:05:23: , Time Taken: 0.001002 seconds
34, Thread ID 21104 at 2024-01-31 22:05:23: , Time Taken: 0.001002 seconds
55, Thread ID 21104 at 2024-01-31 22:05:23: , Time Taken: 0.001002 seconds
89, Thread ID 21104 at 2024-01-31 22:05:23: , Time Taken: 0.001002 seconds
144, Thread ID 21104 at 2024-01-31 22:05:23: , Time Taken: 0.001002 seconds
233, Thread ID 21104 at 2024-01-31 22:05:23: , Time Taken: 0.001002 seconds
377, Thread ID 21104 at 2024-01-31 22:05:23: , Time Taken: 0.001002 seconds

```

Assignment 5

- Consider a scenario where a person visits a supermarket for shopping. S/He purchases various items in different sections such as clothing, grocery, utensils. Write an OpenMP program to process the bill parallelly in each section and display the final amount to be paid by the customer.
- Analyze the time take by sequential and parallel processing.

```
[9]: import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import threading
import time
```

```
[10]: class ProcessingThread(threading.Thread):
    def __init__(self, processing_function, num_items):
        super().__init__()
        self.processing_function = processing_function
        self.num_items = num_items
        self.results = []

    def run(self):
        for _ in range(self.num_items):
            result = self.processing_function()
            self.results.append(result)
```

```
[11]: def process_clothing():
    print("Processing clothing item...")
    time.sleep(0.1)
    return 50
```

```
[12]: def process_grocery():
    print("Processing grocery item...")
    time.sleep(0.1)
    return 50
```

```
[13]: def process_utensils():
    print("Processing utensils item...")
    time.sleep(0.1)
    return 20

[15]: if __name__ == "__main__":
    # For Sequential Processing
    start_time = time.time()

    clothing_cost = sum(process_clothing() for _ in range(10))
    grocery_cost = sum(process_grocery() for _ in range(10))
    utensils_cost = sum(process_utensils() for _ in range(10))

    total_cost = clothing_cost + grocery_cost + utensils_cost
    sequential_time = time.time() - start_time
    print(f"Total amount to be paid (Sequential): ${total_cost:.2f}")
    print(f"Time taken (Sequential): {sequential_time:.2f} seconds\n")

    # For Parallel Processing
    start_time = time.time()

    # Create threads for parallel processing
    num_items = 10
    threads = [
        ProcessingThread(process_clothing, num_items),
        ProcessingThread(process_grocery, num_items),
        ProcessingThread(process_utensils, num_items)
    ]

    for thread in threads:
        thread.start()

    for thread in threads:
        thread.join()

    total_cost_parallel = sum(sum(thread.results) for thread in threads)
    parallel_time = time.time() - start_time
    print(f"Total amount to be paid (Parallel): ${total_cost_parallel:.2f}")
    print(f"Time taken (Parallel): {parallel_time:.2f} seconds")

    labels = ['Sequential', 'Parallel']
    times = [sequential_time, parallel_time]

    plt.bar(labels, times, color=['#40A2E3', '#0D9276'])
    plt.ylabel('Time (seconds)')
    plt.title('Sequential vs Parallel Processing Time Comparison')
    plt.show()
```

Processing clothing item...
Processing grocery item...
Processing utensils item...

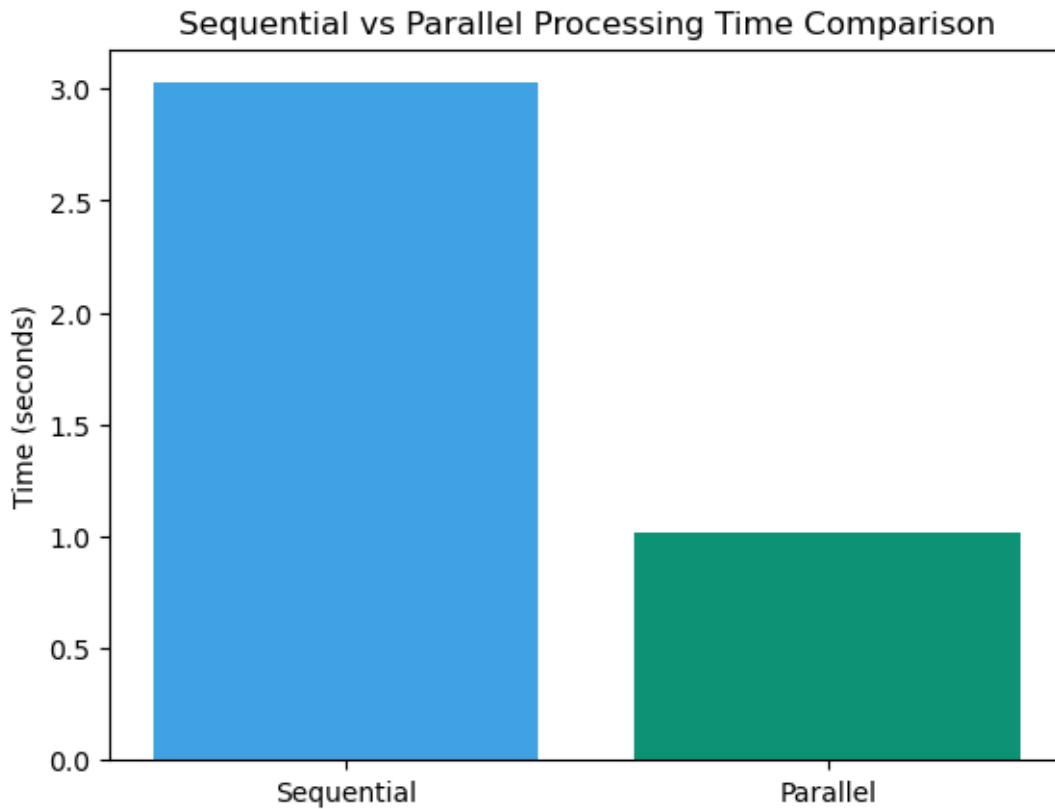
Total amount to be paid (Sequential): \$1200.00

Time taken (Sequential): 3.03 seconds

Processing clothing item...
Processing grocery item...
Processing utensils item...
Processing grocery item...
Processing clothing item...
Processing utensils item...
Processing utensils item...
Processing grocery item...
Processing clothing item...
Processing grocery item...
Processing utensils item...
Processing clothing item...
Processing utensils item...
Processing grocery item...
Processing clothing item...

```
Processing grocery item...
Processing clothing item...
Processing utensils item...
Processing grocery item...
Processing clothing item...
Processing utensils item...
Processing grocery item...
Processing utensils item...
Processing clothing item...
Processing grocery item...Processing utensils item...
```

```
Processing clothing item...
Processing utensils item...
Processing grocery item...
Processing clothing item...
Total amount to be paid (Parallel): $1200.00
Time taken (Parallel): 1.02 seconds
```



Assignment 6

a) Print “Welcome to PDPU from process (processno_totalprocesses)”.

```
[1]: import mpi4py  
from mpi4py import MPI
```

```
[2]: comm = MPI.COMM_WORLD # get the communicator object  
rank = comm.Get_rank() # get the rank of the current process  
name = MPI.Get_processor_name() # get the name of the current processor  
size = comm.Get_size() # get the number of processes  
universe_size = comm.Get_attr(MPI.UNIVERSE_SIZE) # get the expected number of processes
```

```
[3]: print("Welcome to PDPU!")  
print("Process name:", name)  
print("Process id:", rank)  
print("Number of cores:", size)
```

```
Welcome to PDPU!  
Process name: Kavan  
Process id: 0  
Number of cores: 1
```

```
[4]: !mpiexec -n 10 python hpc_mpi.py
```

```
Welcome to PDPU Process name: Kavan Process id: 4 Number of cores: 10  
Welcome to PDPU Process name: Kavan Process id: 8 Number of cores: 10  
Welcome to PDPU Process name: Kavan Process id: 2 Number of cores: 10  
Welcome to PDPU Process name: Kavan Process id: 5 Number of cores: 10  
Welcome to PDPU Process name: Kavan Process id: 7 Number of cores: 10  
Welcome to PDPU Process name: Kavan Process id: 0 Number of cores: 10  
Welcome to PDPU Process name: Kavan Process id: 6 Number of cores: 10  
Welcome to PDPU Process name: Kavan Process id: 3 Number of cores: 10  
Welcome to PDPU Process name: Kavan Process id: 1 Number of cores: 10  
Welcome to PDPU Process name: Kavan Process id: 9 Number of cores: 10
```

b) Apply denoising algorithm to a set of n images with 4 processes. (n=4,8).

```

[57]: import numpy as np
import time

[58]: from scipy.signal import medfilt2d

[59]: from PIL import Image
import matplotlib.pyplot as plt

[60]: def denoise_image(image):
    if len(image.shape) not in [2, 3]:
        raise ValueError("Invalid image format")
    if len(image.shape) == 3:
        denoised_image = np.stack([medfilt2d(channel, kernel_size=3) for
        ↪channel in image.transpose(2, 0, 1)], axis=-1)
    else:
        denoised_image = medfilt2d(image, kernel_size=3)
    return denoised_image

[61]: def chunks(lst, n):
    for i in range(0, len(lst), n):
        yield lst[i:i + n]

[62]: comm = MPI.COMM_WORLD
rank = comm.Get_rank()
size = comm.Get_size()

[63]: image_path = ["noise.png", "noise2.png", "noise3.png", "noise4.jpg"]

[64]: images = [np.array(Image.open(path)) for path in image_path]

[65]: image_chunks = list(chunks(images, len(images) // size))

local_chunks = comm.scatter(image_chunks, root=0)

denoised_chunks = [denoise_image(chunk) for chunk in local_chunks]

all_denoised_chunks = comm.gather(denoised_chunks, root=0)

[66]: if rank == 0:
    # Combine denoised chunks into a single list of denoised images
    denoised_images = [image for sublist in all_denoised_chunks for image in
    ↪sublist]

    # Plot the input and denoised images for each image
    for i, (input_image, denoised_image) in enumerate(zip(images, ↪
    denoised_images)):
        fig, axes = plt.subplots(1, 2, figsize=(12, 5))

```

```

axes[0].imshow(input_image, cmap='gray')
axes[0].set_title(f'Input Image {i+1}')
axes[0].axis('off')
axes[1].imshow(denoised_image, cmap='gray')
axes[1].set_title(f'Denoised Image {i+1}')
axes[1].axis('off')
plt.show()

start_time = MPI.Wtime()
end_time = MPI.Wtime()

print("Process", rank, "took", end_time - start_time, "seconds")

```

Input Image 1



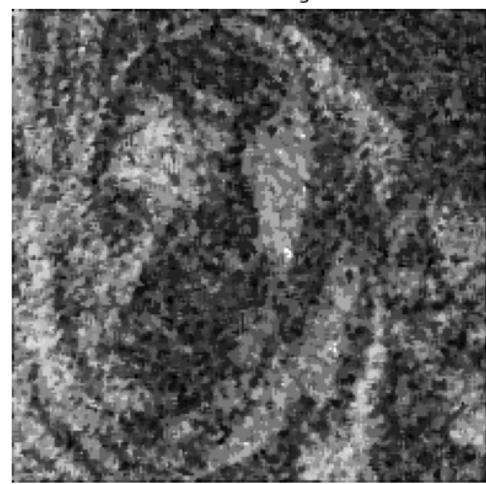
Denoised Image 1



Input Image 2



Denoised Image 2



Input Image 3



Denoised Image 3



Input Image 4



Denoised Image 4



Process 0 took 6.0999998822808266e-05 seconds

[67]: !mpiexec -n 4 python hpc_4_img.py

```
Process 1 took 9.00006853044033e-07 seconds
Process 2 took 1.200009137392044e-06 seconds
Process 3 took 1.300009898841381e-06 seconds
Figure(1200x500)
Figure(1200x500)
Figure(1200x500)
Figure(1200x500)
Process 0 took 8.00006091594696e-07 seconds
```

```
[78]: image_path = ["D:/data set/sports imgs/train/noisy_surf/noisy_001.jpg",
                  "D:/data set/sports imgs/train/noisy_surf/noisy_002.jpg",
                  "D:/data set/sports imgs/train/noisy_surf/noisy_003.jpg",
                  "D:/data set/sports imgs/train/noisy_surf/noisy_004.jpg",
                  "D:/data set/sports imgs/train/noisy_surf/noisy_005.jpg",
                  "D:/data set/sports imgs/train/noisy_surf/noisy_006.jpg",
                  "D:/data set/sports imgs/train/noisy_surf/noisy_007.jpg",
                  "D:/data set/sports imgs/train/noisy_surf/noisy_008.jpg"]

[79]: images = [np.array(Image.open(path)) for path in image_path]

[80]: image_chunks = list(chunks(images, len(images) // size))

local_chunks = comm.scatter(image_chunks, root=0)

denoised_chunks = [denoise_image(chunk) for chunk in local_chunks]

all_denoised_chunks = comm.gather(denoised_chunks, root=0)

[81]: if rank == 0:
    denoised_images = [image for sublist in all_denoised_chunks for image in
    ↪sublist]

    for i, (input_image, denoised_image) in enumerate(zip(images, ↪
    ↪denoised_images)):
        fig, axes = plt.subplots(1, 2, figsize=(12, 5))
        axes[0].imshow(input_image, cmap='gray')
        axes[0].set_title(f'Input Image {i+1}')
        axes[0].axis('off')
        axes[1].imshow(denoised_image, cmap='gray')
        axes[1].set_title(f'Denoised Image {i+1}')
        axes[1].axis('off')
        plt.show()

start_time = MPI.Wtime()
end_time = MPI.Wtime()

print("Process", rank, "took", end_time - start_time, "seconds")
```

Input Image 1



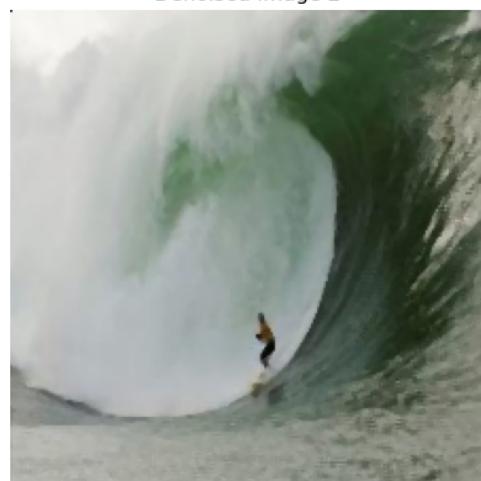
Denoised Image 1



Input Image 2



Denoised Image 2



Input Image 3



Denoised Image 3



Input Image 4



Denoised Image 4



Input Image 5



Denoised Image 5



Input Image 6



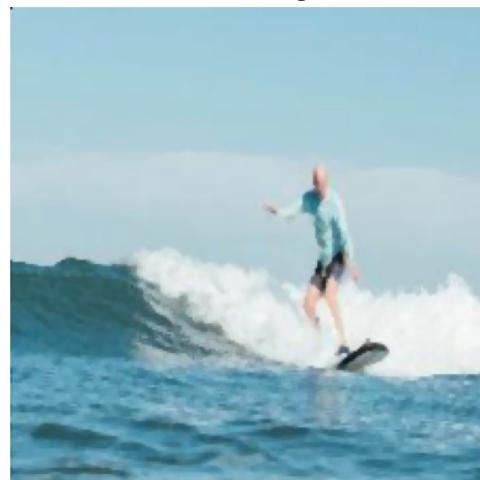
Denoised Image 6



Input Image 7



Denoised Image 7



Input Image 8



Denoised Image 8



Process 0 took 6.360001862049103e-05 seconds

[83]: !mpiexec -n 4 python hpc_8_img.py

Process 1 took 7.00005330145359e-07 seconds

Process 2 took 8.00006091594696e-07 seconds

Process 3 took 9.00006853044033e-07 seconds

Figure(1200x500)

Figure(1200x500)

Figure(1200x500)

Figure(1200x500)

Figure(1200x500)

Figure(1200x500)

Figure(1200x500)

```
Figure(1200x500)
Figure(1200x500)
Process 0 took 9.00006853044033e-07 seconds
```

c) Analyze time taken by serial and openMPI processes.

d) Try for 100 or more number of images.

```
[95]: import os
import cv2

[100]: def add_noise(image):
    noisy_image = image + np.random.normal(loc=0, scale=10, size=image.shape)
    return np.clip(noisy_image, 0, 255).astype(np.uint8)

def denoise(image):
    denoised_image = cv2.fastNlMeansDenoisingColored(image, None, 10, 10, 7, 21)
    return denoised_image

def denoise_images(images, output_folder, rank=None):
    start_time = time.time()
    noisy_output_folder = os.path.join(output_folder, "noisy")
    denoised_output_folder = os.path.join(output_folder, "denoised")
    os.makedirs(noisy_output_folder, exist_ok=True)
    os.makedirs(denoised_output_folder, exist_ok=True)
    for i, image in enumerate(images):
        if image is None:
            print(f"Warning: Image {i} could not be loaded. Skipping.")
            continue
        noisy_image = add_noise(image)
        denoised_image = denoise(noisy_image)
        if rank is None or rank == 0:
            cv2.imwrite(os.path.join(noisy_output_folder, f"noisy_image_{i}.jpg"), noisy_image)
            cv2.imwrite(os.path.join(denoised_output_folder, f"denoised_image_{i}.jpg"), denoised_image)
    end_time = time.time()
    return end_time - start_time

def main():
    comm = MPI.COMM_WORLD
    rank = comm.Get_rank()
    size = comm.Get_size()

    folder_path = folder_path = "D:/data set/sports imgs/train/surfing"
    output_folder = os.path.join(folder_path, "output1")
    image_files = os.listdir(folder_path)
```

```

    images = [cv2.imread(os.path.join(folder_path, file)) for file in
    ↪image_files]

    if rank == 0:
        print(f"Number of images: {len(images)}")

    # Serial denoising
    if rank == 0:
        print("Serial Denoising:")
    comm.Barrier()
    serial_time = denoise_images(images, output_folder, rank)
    if rank == 0:
        print(f"Time taken for serial denoising: {serial_time:.2f} seconds")

    # Parallel denoising
    if rank == 0:
        print("Parallel Denoising:")
    comm.Barrier()
    num_images_per_process = len(images) // size
    start_index = rank * num_images_per_process
    end_index = start_index + num_images_per_process
    parallel_time = denoise_images(images[start_index:end_index], ↪
    ↪output_folder, rank)
    max_parallel_time = comm.reduce(parallel_time, op=MPI.MAX, root=0)
    if rank == 0:
        print(f"Time taken for parallel denoising: {max_parallel_time:.2f} ↪
    ↪seconds")

if __name__ == "__main__":
    main()

```

Number of images: 143
 Serial Denoising:
 Warning: Image 142 could not be loaded. Skipping.
 Time taken for serial denoising: 82.58 seconds
 Parallel Denoising:
 Warning: Image 142 could not be loaded. Skipping.
 Time taken for parallel denoising: 83.79 seconds

[102]: !mpiexec -n 4 python hpc_100_img.py

Warning: Image 142 could not be loaded. Skipping.
 Warning: Image 142 could not be loaded. Skipping.
 Warning: Image 142 could not be loaded. Skipping.
 Number of images: 143
 Serial Denoising:
 Warning: Image 142 could not be loaded. Skipping.
 Time taken for serial denoising: 76.59 seconds

Parallel Denoising:

Time taken for parallel denoising: 21.34 seconds

```
[ WARN:0@0.103] global loadsave.cpp:248 cv::findDecoder imread_('D:/data
set/sports imgs/train/surfing\output1'): can't open/read file: check file
path/integrity
[ WARN:0@0.114] global loadsave.cpp:248 cv::findDecoder imread_('D:/data
set/sports imgs/train/surfing\output1'): can't open/read file: check file
path/integrity
[ WARN:0@0.092] global loadsave.cpp:248 cv::findDecoder imread_('D:/data
set/sports imgs/train/surfing\output1'): can't open/read file: check file
path/integrity
[ WARN:0@0.127] global loadsave.cpp:248 cv::findDecoder imread_('D:/data
set/sports imgs/train/surfing\output1'): can't open/read file: check file
path/integrity
```

[103]: !mpiexec -n 8 python hpc_100_img.py

Warning: Image 142 could not be loaded. Skipping.

```
[ WARN:0@0.093] global loadsave.cpp:248 cv::findDecoder imread_('D:/data
set/sports imgs/train/surfing\output1'): can't open/read file: check file
path/integrity
[ WARN:0@0.120] global loadsave.cpp:248 cv::findDecoder imread_('D:/data
set/sports imgs/train/surfing\output1'): can't open/read file: check file
path/integrity
[ WARN:0@0.121] global loadsave.cpp:248 cv::findDecoder imread_('D:/data
set/sports imgs/train/surfing\output1'): can't open/read file: check file
path/integrity
[ WARN:0@0.148] global loadsave.cpp:248 cv::findDecoder imread_('D:/data
set/sports imgs/train/surfing\output1'): can't open/read file: check file
path/integrity
[ WARN:0@0.136] global loadsave.cpp:248 cv::findDecoder imread_('D:/data
set/sports imgs/train/surfing\output1'): can't open/read file: check file
path/integrity
[ WARN:0@0.168] global loadsave.cpp:248 cv::findDecoder imread_('D:/data
set/sports imgs/train/surfing\output1'): can't open/read file: check file
path/integrity
[ WARN:0@0.188] global loadsave.cpp:248 cv::findDecoder imread_('D:/data
set/sports imgs/train/surfing\output1'): can't open/read file: check file
path/integrity
[ WARN:0@0.113] global loadsave.cpp:248 cv::findDecoder imread_('D:/data
set/sports imgs/train/surfing\output1'): can't open/read file: check file
path/integrity
```

Warning: Image 142 could not be loaded. Skipping.

Number of images: 143

Serial Denoising:

Warning: Image 142 could not be loaded. Skipping.

Time taken for serial denoising: 160.82 seconds

Parallel Denoising:

Time taken for parallel denoising: 20.70 seconds

Assignment 7

1. Write a program to implement arithmetic calculations using MPI processes.
2. Write a program with different processes to apply following functions to an image in parallel.
 - Read an image.
 - Convert above RGB image to grayscale.
 - Find edges in the image.
 - Show the original image.

```
[1]: import mpi4py  
from mpi4py import MPI
```

```
[2]: import numpy as np
```

```
[3]: comm = MPI.COMM_WORLD # get the communicator object  
rank = comm.Get_rank() # get the rank of the current process  
name = MPI.Get_processor_name() # get the name of the current processor  
size = comm.Get_size() # get the number of processes
```

```
[4]: randNum = np.zeros(1)
```

```
[5]: a = 10  
b = 5
```

```
[ ]: if rank == 0:  
    print('rank = ', rank, ', ', a+b)  
if rank == 1:  
    print('rank = ', rank, ', ', a*b)  
if rank == 2:  
    print('rank = ', rank, ', ', a/b)  
if rank == 3:  
    print('rank = ', rank, ', ', a-b)
```

```
[7]: !mpiexec -n 4 python hpc-arith.py
```

```
rank = 0 , addition : 15  
rank = 2 , division : 2.0  
rank = 1 , multiplication : 50  
rank = 3 , subtraction : 5
```

```
[11]: import cv2
```

```
[12]: def read_image(filename):
        image = cv2.imread(filename)
        return image

def convert_to_grayscale(image):
    grayscale_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    return grayscale_image

def find_edges(image):
    edges = cv2.Canny(image, 100, 200)
    return edges

def show_image(image, title="Image"):
    cv2.imshow(title, image)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

```
[13]: filename = "1.jpg"
image = read_image(filename)
```

```
[14]: if rank == 0:
    print('rank = ', rank, ',', 'Read the image')
elif rank == 1:
    grayscale_image = convert_to_grayscale(image)
    print('rank = ', rank, ',', 'Converted RGB image to grayscale')
elif rank == 2:
    edges_image = find_edges(image)
    print('rank = ', rank, ',', 'Found edges in the image')
elif rank == 3:
    show_image(image, title="Original Image")
    print('rank = ', rank, ',', 'Displayed the original image')
    grayscale_image = convert_to_grayscale(image)
    show_image(grayscale_image, title="Grayscale Image")
    print('rank = ', rank, ',', 'Displayed the grayscale image')
    edges_image = find_edges(image)
    show_image(edges_image, title="Edges Image")
    print('rank = ', rank, ',', 'Displayed the edges image')
```

rank = 0 , Read the image

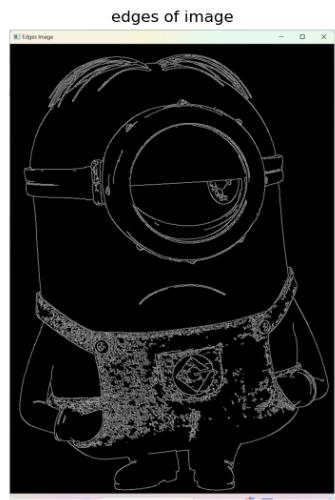
```
[15]: !mpiexec -n 4 python hpc-7(2).py
```

rank = 0 , Read the image
rank = 1 , Converted RGB image to grayscale
rank = 2 , Found edges in the image
rank = 3 , Displayed the original image

```
rank = 3 , Displayed the grayscale image  
rank = 3 , Displayed the edges image
```

```
[16]: import matplotlib.pyplot as plt
```

```
[19]: # Load the images  
image1 = cv2.imread("hpc-7-2-2.png")  
image2 = cv2.imread("hpc-7-2-3.png")  
image3 = cv2.imread("hpc-7-2-1.png")  
  
# Convert BGR to RGB (Matplotlib uses RGB)  
image1_rgb = cv2.cvtColor(image1, cv2.COLOR_BGR2RGB)  
image2_rgb = cv2.cvtColor(image2, cv2.COLOR_BGR2RGB)  
image3_rgb = cv2.cvtColor(image3, cv2.COLOR_BGR2RGB)  
  
# Display the images  
plt.figure(figsize=(15, 10))  
  
plt.subplot(1, 3, 1)  
plt.imshow(image1_rgb)  
plt.axis('off')  
plt.title('grayscaled image')  
  
plt.subplot(1, 3, 2)  
plt.imshow(image2_rgb)  
plt.axis('off')  
plt.title('edges of image')  
  
plt.subplot(1, 3, 3)  
plt.imshow(image3_rgb)  
plt.axis('off')  
plt.title('original image')  
  
plt.show()
```



Assignment 8

1. Write a program to pass message from one process to another and print output.
 - In synchronous communication
 - In asynchronous communication. Show using overlapping of task in non-blocking mode.

```
[1]: import mpi4py  
from mpi4py import MPI
```

```
[2]: import numpy as np
```

```
[3]: comm = MPI.COMM_WORLD # get the communicator object  
rank = comm.Get_rank() # get the rank of the current process  
name = MPI.Get_processor_name() # get the name of the current processor  
size = comm.Get_size() # get the number of processes
```

```
[4]: randNum = np.zeros(1)
```

```
[ ]: if rank == 0:  
    message = "Hello from process 0"  
    comm.send(message, dest=1)  
  
    received_message = comm.recv(source=1)  
    print(f"Process 0 received message: {received_message}")  
  
elif rank == 1:  
    received_message = comm.recv(source=0)  
    print(f"Process 1 received message: {received_message}")  
  
    reply = "Hello from process 1"  
    comm.send(reply, dest=0)
```

```
[5]: !mpiexec -n 2 python hpc-12-2.py
```

```
Process 1 received message: Hello from process 0  
Process 0 received message: Hello from process 1
```

```
[ ]: if rank == 0:  
    message = "Hello from process 0 (Async)"  
    req_send = comm.isend(message, dest=1)  
    print(f"Process {rank} sent message: {message}")  
    time.sleep(1)  
    req_send.wait()  
elif rank == 1:  
    req_recv = comm.irecv(source=0)  
    time.sleep(0.5)  
    print(f"Process {rank} waiting to receive message...")  
    received_message = req_recv.wait()  
    print(f"Process {rank} received message: {received_message}")
```

```
[8]: !mpiexec -n 2 python hpc-async.py
```

```
Process 0 sent message: Hello from process 0 (Async)  
Process 1 waiting to receive message..  
Process 1 received message: Hello from process 0 (Async)
```

Assignment 9

1. Calculate Pi value using openMPI send and receive messages for atleast 35-40 terms.
2. Change the value on n as 2, 4, 8, 16.
3. Analyze the performance improvement using number of processes.

```
[1]: import mpi4py
from mpi4py import MPI

[2]: import time

[3]: TERMS = 40

[4]: def calculate_pi(rank, num_processes, terms):
    partial_sum = 0.0
    for i in range(rank, terms, num_processes):
        if i % 2 == 0:
            partial_sum += 1.0 / (2 * i + 1)
        else:
            partial_sum -= 1.0 / (2 * i + 1)
    return partial_sum * 4

[5]: if __name__ == "__main__":
    comm = MPI.COMM_WORLD
    rank = comm.Get_rank()
    size = comm.Get_size()

    terms = 40

    start_time = time.time()

    partial_pi = calculate_pi(rank, size, terms)
    print(f"calculated Rank {rank} : Pi = {partial_pi}, Time = {time.time() - start_time}")

    if rank == 0:
        total_pi = partial_pi
```

```

        for i in range(1, size):
            partial_result, partial_time = comm.recv(source=i)
            total_pi += partial_result
            print(f"Received from Rank {i}: Pi = {partial_result}, Time = {partial_time}")

        print("Number of processes:", size)
        print("Estimated Pi:", total_pi)
        print("Execution time:", time.time() - start_time, "seconds")
    else:
        print(f"Sending from Rank {rank}: Pi = {partial_pi}, Time = {time.time() - start_time}")
    comm.send((partial_pi, time.time() - start_time), dest=0)

```

calculated Rank 0 : Pi = 3.116596556793833, Time = 0.0
Number of processes: 1
Estimated Pi: 3.116596556793833
Execution time: 0.0 seconds

[6]: !mpiexec -n 2 python hpc-as9.py

calculated Rank 1 : Pi = -4.0941172405577415, Time = 0.0
Sending from Rank 1: Pi = -4.0941172405577415, Time = 0.0
calculated Rank 0 : Pi = 7.210713797351573, Time = 0.0
Received from Rank 1: Pi = -4.0941172405577415, Time = 0.0
Number of processes: 2
Estimated Pi: 3.116596556793832
Execution time: 0.0030989646911621094 seconds

[7]: !mpiexec -n 4 python hpc-as9.py

calculated Rank 2 : Pi = 1.8840610646940703, Time = 0.0
Sending from Rank 2: Pi = 1.8840610646940703, Time = 0.0
calculated Rank 3 : Pi = -1.5719013009615963, Time = 0.0
Sending from Rank 3: Pi = -1.5719013009615963, Time = 0.0
calculated Rank 1 : Pi = -2.5222159395961445, Time = 0.0
Sending from Rank 1: Pi = -2.5222159395961445, Time = 0.0
calculated Rank 0 : Pi = 5.326652732657504, Time = 0.0
Received from Rank 1: Pi = -2.5222159395961445, Time = 0.0
Received from Rank 2: Pi = 1.8840610646940703, Time = 0.0
Received from Rank 3: Pi = -1.5719013009615963, Time = 0.0
Number of processes: 4
Estimated Pi: 3.1165965567938327
Execution time: 0.005960226058959961 seconds

[8]: !mpiexec -n 8 python hpc-as9.py

calculated Rank 7 : Pi = -0.5949302825943747, Time = 0.0
Sending from Rank 7: Pi = -0.5949302825943747, Time = 0.0

```

calculated Rank 3 : Pi = -0.9769710183672217, Time = 0.0
Sending from Rank 3: Pi = -0.9769710183672217, Time = 0.0
calculated Rank 5 : Pi = -0.7259377111012902, Time = 0.0
Sending from Rank 5: Pi = -0.7259377111012902, Time = 0.0
calculated Rank 2 : Pi = 1.2320270111902598, Time = 0.0
Sending from Rank 2: Pi = 1.2320270111902598, Time = 0.0
calculated Rank 4 : Pi = 0.826975379198637, Time = 0.0
Sending from Rank 4: Pi = 0.826975379198637, Time = 0.0
calculated Rank 1 : Pi = -1.7962782284948542, Time = 0.0
Sending from Rank 1: Pi = -1.7962782284948542, Time = 0.0
calculated Rank 6 : Pi = 0.6520340535038104, Time = 0.0
Sending from Rank 6: Pi = 0.6520340535038104, Time = 0.0
calculated Rank 0 : Pi = 4.499677353458866, Time = 0.0
Received from Rank 1: Pi = -1.7962782284948542, Time = 0.0
Received from Rank 2: Pi = 1.2320270111902598, Time = 0.0
Received from Rank 3: Pi = -0.9769710183672217, Time = 0.0
Received from Rank 4: Pi = 0.826975379198637, Time = 0.0
Received from Rank 5: Pi = -0.7259377111012902, Time = 0.0
Received from Rank 6: Pi = 0.6520340535038104, Time = 0.0
Received from Rank 7: Pi = -0.5949302825943747, Time = 0.0
Number of processes: 8
Estimated Pi: 3.116596556793832
Execution time: 0.005739927291870117 seconds

```

[9]: !mpiexec -n 16 python hpc-as9.py

```

calculated Rank 14 : Pi = 0.20350480497456191, Time = 0.0
Sending from Rank 14: Pi = 0.20350480497456191, Time = 0.0
calculated Rank 7 : Pi = -0.40240596103779513, Time = 0.0
Sending from Rank 7: Pi = -0.40240596103779513, Time = 0.0
calculated Rank 11 : Pi = -0.2466403162055336, Time = 0.0
Sending from Rank 11: Pi = -0.2466403162055336, Time = 0.0
calculated Rank 8 : Pi = 0.3169267707082833, Time = 0.0
Sending from Rank 8: Pi = 0.3169267707082833, Time = 0.0
calculated Rank 3 : Pi = -0.730330702161688, Time = 0.0
Sending from Rank 3: Pi = -0.730330702161688, Time = 0.0
calculated Rank 5 : Pi = -0.5099929527836504, Time = 0.0
Sending from Rank 5: Pi = -0.5099929527836504, Time = 0.0
calculated Rank 1 : Pi = -1.507320540156361, Time = 0.0
Sending from Rank 1: Pi = -1.507320540156361, Time = 0.0
calculated Rank 15 : Pi = -0.19252432155657961, Time = 0.0
Sending from Rank 15: Pi = -0.19252432155657961, Time = 0.0
calculated Rank 2 : Pi = 0.9660791226008618, Time = 0.0
Sending from Rank 2: Pi = 0.9660791226008618, Time = 0.0
calculated Rank 10 : Pi = 0.265947888589398, Time = 0.0
Sending from Rank 10: Pi = 0.265947888589398, Time = 0.0
calculated Rank 9 : Pi = -0.2889576883384933, Time = 0.0
Sending from Rank 9: Pi = -0.2889576883384933, Time = 0.0

```

```

calculated Rank 6 : Pi = 0.4485292485292486, Time = 0.0
Sending from Rank 6: Pi = 0.4485292485292486, Time = 0.0
calculated Rank 4 : Pi = 0.5967999406021457, Time = 0.0
Sending from Rank 4: Pi = 0.5967999406021457, Time = 0.0
calculated Rank 13 : Pi = -0.21594475831763965, Time = 0.0
Sending from Rank 13: Pi = -0.21594475831763965, Time = 0.0
calculated Rank 12 : Pi = 0.23017543859649123, Time = 0.0
Sending from Rank 12: Pi = 0.23017543859649123, Time = 0.0
calculated Rank 0 : Pi = 4.182750582750582, Time = 0.0
Received from Rank 1: Pi = -1.507320540156361, Time = 0.0
Received from Rank 2: Pi = 0.9660791226008618, Time = 0.0
Received from Rank 3: Pi = -0.730330702161688, Time = 0.0
Received from Rank 4: Pi = 0.5967999406021457, Time = 0.0
Received from Rank 5: Pi = -0.5099929527836504, Time = 0.0
Received from Rank 6: Pi = 0.4485292485292486, Time = 0.0
Received from Rank 7: Pi = -0.40240596103779513, Time = 0.0
Received from Rank 8: Pi = 0.3169267707082833, Time = 0.0
Received from Rank 9: Pi = -0.2889576883384933, Time = 0.0
Received from Rank 10: Pi = 0.265947888589398, Time = 0.0
Received from Rank 11: Pi = -0.2466403162055336, Time = 0.0
Received from Rank 12: Pi = 0.23017543859649123, Time = 0.0
Received from Rank 13: Pi = -0.21594475831763965, Time = 0.0
Received from Rank 14: Pi = 0.20350480497456191, Time = 0.0
Received from Rank 15: Pi = -0.19252432155657961, Time = 0.0
Number of processes: 16
Estimated Pi: 3.1165965567938323
Execution time: 0.006808042526245117 seconds

```

```
[10]: total_time = [0.0030989646911621094 , 0.005960226058959961 , 0.
      ↪005739927291870117 , 0.006808042526245117]
total_time
```

```
[10]: [0.0030989646911621094,
      0.005960226058959961,
      0.005739927291870117,
      0.006808042526245117]
```

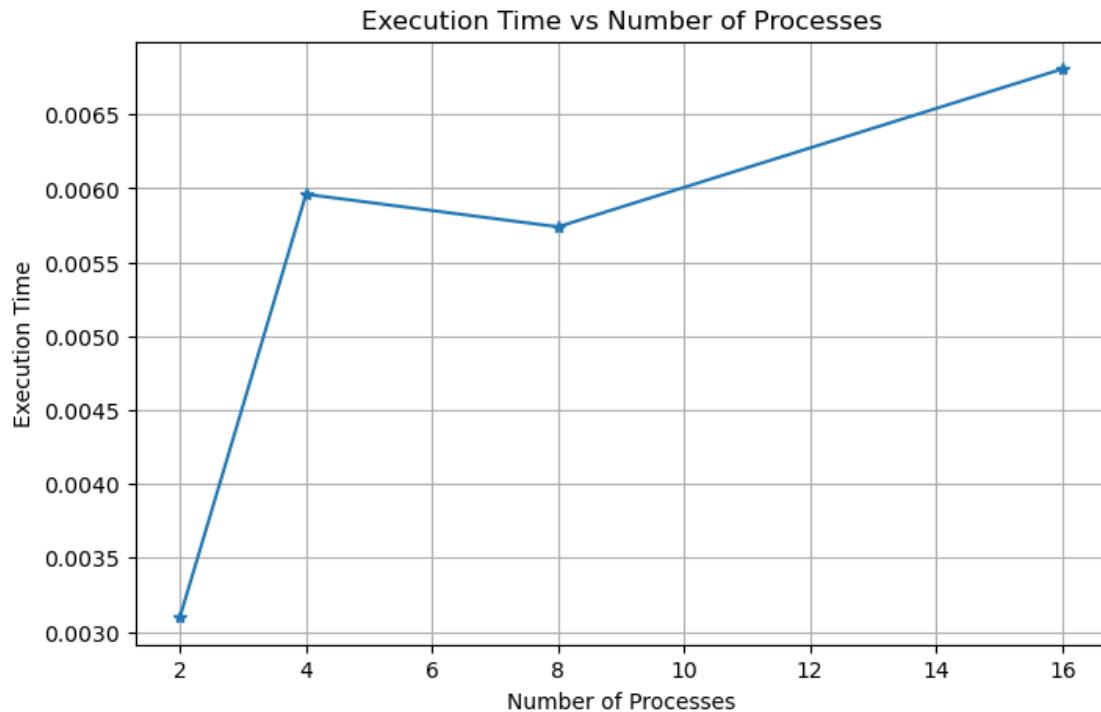
```
[11]: ranks = [2,4,8,16]
ranks
```

```
[11]: [2, 4, 8, 16]
```

```
[13]: import matplotlib.pyplot as plt
```

```
[14]: plt.figure(figsize=(8, 5))
plt.plot(ranks, total_time, marker='*')
plt.title('Execution Time vs Number of Processes')
plt.xlabel('Number of Processes')
```

```
plt.ylabel('Execution Time')
plt.grid(True)
plt.show()
```



Assignment 10

Write a program to show collective communication by taking suitable example such that computing average of n numbers or computing sum or product of two matrices... :

- Broadcast function
- Scatter function
- Gather function

[1]:

```
from mpi4py import MPI
import numpy as np
```

[2]:

```
comm = MPI.COMM_WORLD
rank = comm.Get_rank()
size = comm.Get_size()
```

[3]:

```
n = 10
local_sum = np.random.randint(0, 100, n)

local_sum_total = np.sum(local_sum)

global_sum = np.array(0, dtype='i')
comm.Reduce(local_sum_total, global_sum, op=MPI.SUM, root=0)

if rank == 0:
    print("Global sum:", global_sum)
```

Global sum: 533

[6]:

```
!mpiexec -n 10 python bcast.py
```

```
Rank 0 is broadcasting
Process : 0 received data : 66
Process : 8 , is waiting to receive data from Rank 0
Process : 8 received data : 66
Process : 9 , is waiting to receive data from Rank 0
Process : 9 received data : 66
Process : 1 , is waiting to receive data from Rank 0
Process : 1 received data : 66
Process : 2 , is waiting to receive data from Rank 0
```

```
Process : 2 received data : 66
Process : 3 , is waiting to receive data from Rank 0
Process : 3 received data : 66
Process : 4 , is waiting to receive data from Rank 0
Process : 4 received data : 66
Process : 5 , is waiting to receive data from Rank 0
Process : 5 received data : 66
Process : 6 , is waiting to receive data from Rank 0
Process : 6 received data : 66
Process : 7 , is waiting to receive data from Rank 0
Process : 7 received data : 66
```

```
[7]: if rank == 0:
    print("Rank 0 is scattering")
else:
    print("Process : ", rank, " , is waiting to receive scattered data from\u2192Rank 0")

if rank == 0:
    send_data = np.arange(size) * 10
else:
    send_data = None

recv_data = np.empty(1, dtype=int)
comm.Scatter(send_data, recv_data, root=0)

print("Process : ", rank, "received data : ", recv_data[0])
```

```
Rank 0 is scattering
Process : 0 received data : 0
```

```
[8]: !mpiexec -n 10 python scatter.py
```

```
Process : 8 , is waiting to receive scattered data from Rank 0
Process : 8 received data : 80
Rank 0 is scattering
Process : 0 received data : 0
Process : 1 , is waiting to receive scattered data from Rank 0
Process : 1 received data : 10
Process : 4 , is waiting to receive scattered data from Rank 0
Process : 4 received data : 40
Process : 2 , is waiting to receive scattered data from Rank 0
Process : 2 received data : 20
Process : 5 , is waiting to receive scattered data from Rank 0
Process : 5 received data : 50
Process : 6 , is waiting to receive scattered data from Rank 0
Process : 6 received data : 60
Process : 3 , is waiting to receive scattered data from Rank 0
Process : 3 received data : 30
```

```
Process : 9 , is waiting to receive scattered data from Rank 0
Process : 9 received data : 90
Process : 7 , is waiting to receive scattered data from Rank 0
Process : 7 received data : 70
```

```
[9]: local_sum = np.random.randint(0, 100)

if rank == 0:
    print("Rank 0 is gathering")

global_sums = None
if rank == 0:
    global_sums = np.empty(size, dtype=int)
comm.Gather(np.array(local_sum, dtype=int), global_sums, root=0)

if rank == 0:
    print("Rank 0 gathered the following local sums : ", global_sums)
```

```
Rank 0 is gathering
Rank 0 gathered the following local sums : [77]
```

```
[10]: !mpiexec -n 10 python gather.py
```

```
Rank 0 is gathering
Rank 0 gathered the following local sums : [25 4 22 96 94 95 54 17 40 76]
```

Assignment 11

Canon's Matrix Multiplication

1. Describe Canon's Matrix Multiplication algorithm.

A. Cannon's algorithm is a distributed matrix multiplication algorithm that works particularly well with two-dimensional meshes. Lynn Elliot Cannon described it for the first time in 1969.

Algorithm Overview:

- * When multiplying two $n \times n$ matrices A and B, we need $n \times n$ processing nodes arranged in a 2D.
- * Each processing node (PE) performs the following steps:
 1. Initialize: $k = (i + j) \bmod n$ (to ensure different processors access distinct data).
 2. Compute: $c[i][j] += a[i][k] \times b[k][j]$ concurrently.
 3. Communicate:
 - * Send a to PE $(i, (j + N - 1) \bmod n)$.
 - * Send b to PE $((i + N - 1) \bmod n, j)$.
 - * Receive a' from PE $(i, (j + 1) \bmod n)$.
 - * Receive b' from PE $((i + 1) \bmod n, j)$.
 4. Update: $a = a'$ and $b = b'$.
- * Repeat the above steps for n iterations, ensuring that each processor accesses different matrix elements.
- * The storage requirements remain constant and independent of the number of processors.

2. Implement Canon's Matrix Multiplication using collective communication.

3. Analyze the efficiency of the code.

[1]:
from mpi4py import MPI
import numpy as np

[2]:
comm = MPI.COMM_WORLD
rank = comm.Get_rank()
size = comm.Get_size()

[3]:
import time
import matplotlib.pyplot as plt

[4]:
N = 2000

```
[5]: if N % size != 0:
    raise ValueError("Matrix size N must be divisible by the number of processes (size)")

block_size = N // size

print(f"Rank {rank}: Starting execution")

if rank == 0:
    print(f"Rank {rank}: Generating matrices A and B")
    A = np.random.randint(0, 10, (N, N))
    B = np.random.randint(0, 10, (N, N))
    print(f"Rank {rank}: Matrices A and B generated")
else:
    A = None
    B = None

print(f"Rank {rank}: Broadcasting matrices A and B")
```

Rank 0: Starting execution
 Rank 0: Generating matrices A and B
 Rank 0: Matrices A and B generated
 Rank 0: Broadcasting matrices A and B

```
[6]: start_time = time.time()
A = comm.bcast(A, root=0)
B = comm.bcast(B, root=0)
end_time = time.time()
print(f"Rank {rank}: Matrices A and B broadcasted")
```

Rank 0: Matrices A and B broadcasted

```
[7]: A_rows = np.zeros((block_size, N), dtype=int)
comm.Scatter(A, A_rows, root=0)

start_time_multiplication = time.time()
C_rows = np.dot(A_rows, B)
end_time_multiplication = time.time()

C = None
if rank == 0:
    C = np.zeros((N, N), dtype=int)

comm.Gather(C_rows, C, root=0)

if rank == 0:
    print("Resultant Matrix C:")
    print(C)
```

```

    print("Broadcasting time:", end_time - start_time, "seconds")
    print("Matrix multiplication time:", end_time_multiplication -_
         start_time_multiplication, "seconds")

```

Resultant Matrix C:

```

[[40747 41349 41354 ... 41260 41169 39328]
 [39955 41279 41428 ... 40482 42471 40155]
 [40196 41163 40919 ... 40635 39846 39888]
 ...
 [40674 41117 40256 ... 40439 39831 39460]
 [39447 40800 40591 ... 40467 40723 40124]
 [40330 40734 40834 ... 41727 41303 40117]]

```

Broadcasting time: 0.039728403091430664 seconds
Matrix multiplication time: 20.929131031036377 seconds

[8]: !mpiexec -n 4 python mpi_scatter_gather.py

```

Rank 3: Starting execution
Rank 3: Broadcasting matrices A and B
Rank 3: Matrices A and B broadcasted
Rank 1: Starting execution
Rank 1: Broadcasting matrices A and B
Rank 1: Matrices A and B broadcasted
Rank 2: Starting execution
Rank 2: Broadcasting matrices A and B
Rank 2: Matrices A and B broadcasted
Rank 0: Starting execution
Rank 0: Generating matrices A and B
Rank 0: Matrices A and B generated
Rank 0: Broadcasting matrices A and B
Rank 0: Matrices A and B broadcasted
Resultant Matrix C:
[[41048 39762 40853 ... 41543 41611 39830]
 [40264 38648 41718 ... 40252 40400 39149]
 [41054 40522 41801 ... 41724 41400 41007]
 ...
 [41452 39388 41118 ... 41178 40805 40147]
 [39720 38856 40107 ... 40178 40473 39127]
 [40218 38532 40695 ... 40379 40558 39088]]

```

Broadcasting time: 0.03373289108276367 seconds
Matrix multiplication time: 4.518505096435547 seconds

[9]: !mpiexec -n 8 python mpi_scatter_gather.py

```

Rank 5: Starting execution
Rank 5: Broadcasting matrices A and B
Rank 5: Matrices A and B broadcasted
Rank 7: Starting execution
Rank 7: Broadcasting matrices A and B

```

```
Rank 7: Matrices A and B broadcasted
Rank 6: Starting execution
Rank 6: Broadcasting matrices A and B
Rank 6: Matrices A and B broadcasted
Rank 1: Starting execution
Rank 1: Broadcasting matrices A and B
Rank 1: Matrices A and B broadcasted
Rank 3: Starting execution
Rank 3: Broadcasting matrices A and B
Rank 3: Matrices A and B broadcasted
Rank 2: Starting execution
Rank 2: Broadcasting matrices A and B
Rank 2: Matrices A and B broadcasted
Rank 4: Starting execution
Rank 4: Broadcasting matrices A and B
Rank 4: Matrices A and B broadcasted
Rank 0: Starting execution
Rank 0: Generating matrices A and B
Rank 0: Matrices A and B generated
Rank 0: Broadcasting matrices A and B
Rank 0: Matrices A and B broadcasted
Resultant Matrix C:
[[39642 40083 40950 ... 41662 41105 40729]
 [40628 40791 40643 ... 41148 41905 39657]
 [40344 41142 40593 ... 42146 42763 40476]
 ...
 [39234 39860 40460 ... 40807 40930 40160]
 [38715 39396 39736 ... 41051 39867 39587]
 [40204 40537 40572 ... 41915 41841 40698]]
Broadcasting time: 0.066436767578125 seconds
Matrix multiplication time: 2.638921022415161 seconds
```

```
[10]: !mpiexec -n 16 python mpi_scatter_gather.py
```

```
Rank 5: Starting execution
Rank 5: Broadcasting matrices A and B
Rank 5: Matrices A and B broadcasted
Rank 15: Starting execution
Rank 15: Broadcasting matrices A and B
Rank 15: Matrices A and B broadcasted
Rank 9: Starting execution
Rank 9: Broadcasting matrices A and B
Rank 9: Matrices A and B broadcasted
Rank 11: Starting execution
Rank 11: Broadcasting matrices A and B
Rank 11: Matrices A and B broadcasted
Rank 10: Starting execution
Rank 10: Broadcasting matrices A and B
```

```
Rank 10: Matrices A and B broadcasted
Rank 1: Starting execution
Rank 1: Broadcasting matrices A and B
Rank 1: Matrices A and B broadcasted
Rank 3: Starting execution
Rank 3: Broadcasting matrices A and B
Rank 3: Matrices A and B broadcasted
Rank 2: Starting execution
Rank 2: Broadcasting matrices A and B
Rank 2: Matrices A and B broadcasted
Rank 7: Starting execution
Rank 7: Broadcasting matrices A and B
Rank 7: Matrices A and B broadcasted
Rank 6: Starting execution
Rank 6: Broadcasting matrices A and B
Rank 6: Matrices A and B broadcasted
Rank 4: Starting execution
Rank 4: Broadcasting matrices A and B
Rank 4: Matrices A and B broadcasted
Rank 13: Starting execution
Rank 13: Broadcasting matrices A and B
Rank 13: Matrices A and B broadcasted
Rank 14: Starting execution
Rank 14: Broadcasting matrices A and B
Rank 14: Matrices A and B broadcasted
Rank 12: Starting execution
Rank 12: Broadcasting matrices A and B
Rank 12: Matrices A and B broadcasted
Rank 8: Starting execution
Rank 8: Broadcasting matrices A and B
Rank 8: Matrices A and B broadcasted
Rank 0: Starting execution
Rank 0: Generating matrices A and B
Rank 0: Matrices A and B generated
Rank 0: Broadcasting matrices A and B
Rank 0: Matrices A and B broadcasted
Resultant Matrix C:
[[39895 40041 38242 ... 39939 39545 40169]
 [39345 39674 38612 ... 39800 40560 38977]
 [40013 39937 38095 ... 40518 41096 40771]
 ...
 [40008 39913 39012 ... 39942 41684 39822]
 [41780 41378 40286 ... 41451 42157 41894]
 [40301 40239 39599 ... 39864 40844 40837]]
Broadcasting time: 0.22203373908996582 seconds
Matrix multiplication time: 3.5083491802215576 seconds
```

```
[11]: N = 2000
if N % size != 0:
    raise ValueError("Matrix size N must be divisible by the number of processes (size)")

block_size = N // size

print(f"Rank {rank}: Starting execution")

if rank == 0:
    print(f"Rank {rank}: Generating matrices A and B")
    A = np.random.randint(0, 10, (N, N))
    B = np.random.randint(0, 10, (N, N))
    print(f"Rank {rank}: Matrices A and B generated")
else:
    A = None
    B = None

print(f"Rank {rank}: Broadcasting matrices A and B")
```

Rank 0: Starting execution
 Rank 0: Generating matrices A and B
 Rank 0: Matrices A and B generated
 Rank 0: Broadcasting matrices A and B

```
[12]: start_time = time.time()
A = comm.bcast(A, root=0)
B = comm.bcast(B, root=0)
end_time = time.time()
print(f"Rank {rank}: Matrices A and B broadcasted")

A_rows = np.zeros((block_size, N), dtype=int)
comm.Scatter(A, A_rows, root=0)

start_time_multiplication = time.time()
C_rows = np.dot(A_rows, B)
end_time_multiplication = time.time()

start_time_gather = time.time()
C_all = np.zeros((N, N), dtype=int)
comm.Allgather(C_rows, C_all)
end_time_gather = time.time()

if rank == 0:
    print("Resultant Matrix C:")
    print(C_all)
    print("Broadcasting time:", end_time - start_time, "seconds")
```

```

    print("Gathering time:", end_time_gather - start_time_gather, "seconds")
    print("Matrix multiplication time:", end_time_multiplication -_
         start_time_multiplication, "seconds")

```

Rank 0: Matrices A and B broadcasted
Resultant Matrix C:
[[40262 41419 41149 ... 40627 41185 39118]
 [40512 40635 39829 ... 40602 41313 39061]
 [41364 40627 40990 ... 40016 41141 40741]
 ...
 [40798 40859 41008 ... 39674 40921 40082]
 [40201 40859 40919 ... 40100 40550 39726]
 [41429 40674 40897 ... 39753 41346 39610]]
Broadcasting time: 0.036681175231933594 seconds
Gathering time: 0.007696866989135742 seconds
Matrix multiplication time: 21.168152570724487 seconds

[13]: !mpiexec -n 4 python MPI_Allgather.py

Rank 3: Starting execution
Rank 3: Broadcasting matrices A and B
Rank 3: Matrices A and B broadcasted
Rank 1: Starting execution
Rank 1: Broadcasting matrices A and B
Rank 1: Matrices A and B broadcasted
Rank 2: Starting execution
Rank 2: Broadcasting matrices A and B
Rank 2: Matrices A and B broadcasted
Rank 0: Starting execution
Rank 0: Generating matrices A and B
Rank 0: Matrices A and B generated
Rank 0: Broadcasting matrices A and B
Rank 0: Matrices A and B broadcasted
Resultant Matrix C:
[[39268 39927 39869 ... 39620 39577 39546]
 [40260 41715 40449 ... 41332 40204 40137]
 [39132 40005 39775 ... 40137 39549 40140]
 ...
 [40205 40717 40398 ... 40964 40263 40094]
 [40544 41437 40248 ... 42028 40126 40540]
 [40018 40157 40004 ... 41190 39445 40415]]
Broadcasting time: 0.031401872634887695 seconds
Gathering time: 0.011379480361938477 seconds
Matrix multiplication time: 4.781134605407715 seconds

[14]: !mpiexec -n 8 python MPI_Allgather.py

Rank 3: Starting execution
Rank 3: Broadcasting matrices A and B

```

Rank 3: Matrices A and B broadcasted
Rank 2: Starting execution
Rank 2: Broadcasting matrices A and B
Rank 2: Matrices A and B broadcasted
Rank 4: Starting execution
Rank 4: Broadcasting matrices A and B
Rank 4: Matrices A and B broadcasted
Rank 7: Starting execution
Rank 7: Broadcasting matrices A and B
Rank 7: Matrices A and B broadcasted
Rank 1: Starting execution
Rank 1: Broadcasting matrices A and B
Rank 1: Matrices A and B broadcasted
Rank 0: Starting execution
Rank 0: Generating matrices A and B
Rank 0: Matrices A and B generated
Rank 0: Broadcasting matrices A and B
Rank 0: Matrices A and B broadcasted
Resultant Matrix C:
[[40351 39324 40644 ... 40178 40722 38738]
 [41633 41139 41735 ... 41205 41598 40359]
 [42168 40904 41892 ... 40341 42459 40579]
 ...
 [41025 40337 40735 ... 41337 41649 40040]
 [39452 38465 39429 ... 39067 39286 38344]
 [41148 40563 41671 ... 41142 41364 40395]]
Broadcasting time: 0.06987953186035156 seconds
Gathering time: 0.05123305320739746 seconds
Matrix multiplication time: 2.8613476753234863 seconds
Rank 5: Starting execution
Rank 5: Broadcasting matrices A and B
Rank 5: Matrices A and B broadcasted
Rank 6: Starting execution
Rank 6: Broadcasting matrices A and B
Rank 6: Matrices A and B broadcasted

```

[15]: !mpiexec -n 16 python MPI_Allgather.py

```

Rank 9: Starting execution
Rank 9: Broadcasting matrices A and B
Rank 9: Matrices A and B broadcasted
Rank 10: Starting execution
Rank 10: Broadcasting matrices A and B
Rank 10: Matrices A and B broadcasted
Rank 7: Starting execution
Rank 7: Broadcasting matrices A and B
Rank 7: Matrices A and B broadcasted
Rank 8: Starting execution

```

```
Rank 8: Broadcasting matrices A and B
Rank 8: Matrices A and B broadcasted
Rank 5: Starting execution
Rank 5: Broadcasting matrices A and B
Rank 5: Matrices A and B broadcasted
Rank 6: Starting execution
Rank 6: Broadcasting matrices A and B
Rank 6: Matrices A and B broadcasted
Rank 11: Starting execution
Rank 11: Broadcasting matrices A and B
Rank 11: Matrices A and B broadcasted
Rank 13: Starting execution
Rank 13: Broadcasting matrices A and B
Rank 13: Matrices A and B broadcasted
Rank 12: Starting execution
Rank 12: Broadcasting matrices A and B
Rank 12: Matrices A and B broadcasted
Rank 0: Starting execution
Rank 0: Generating matrices A and B
Rank 0: Matrices A and B generated
Rank 0: Broadcasting matrices A and B
Rank 0: Matrices A and B broadcasted
Resultant Matrix C:
[[40098 40513 39370 ... 40675 40116 40372]
 [40373 40444 39136 ... 40684 39919 40297]
 [40017 40324 39607 ... 39830 40199 39921]
 ...
 [40336 40603 39674 ... 40866 39942 40451]
 [40586 40233 39441 ... 40566 39837 40573]
 [40127 41018 40006 ... 41142 40468 40251]]
Broadcasting time: 0.11317896842956543 seconds
Gathering time: 0.7809598445892334 seconds
Matrix multiplication time: 2.607423782348633 seconds
Rank 14: Starting execution
Rank 14: Broadcasting matrices A and B
Rank 14: Matrices A and B broadcasted
Rank 15: Starting execution
Rank 15: Broadcasting matrices A and B
Rank 15: Matrices A and B broadcasted
Rank 1: Starting execution
Rank 1: Broadcasting matrices A and B
Rank 1: Matrices A and B broadcasted
Rank 2: Starting execution
Rank 2: Broadcasting matrices A and B
Rank 2: Matrices A and B broadcasted
Rank 4: Starting execution
Rank 4: Broadcasting matrices A and B
Rank 4: Matrices A and B broadcasted
```

```
Rank 3: Starting execution
Rank 3: Broadcasting matrices A and B
Rank 3: Matrices A and B broadcasted
```

```
[22]: scatter_gather_processes = [4, 8, 16]
scatter_gather_broadcasting_time = [0.03373289108276367, 0.066436767578125, 0.
    ↪22203373908996582]
scatter_gather_multiplication_time = [4.518505096435547, 2.638921022415161, 3.
    ↪5083491802215576]

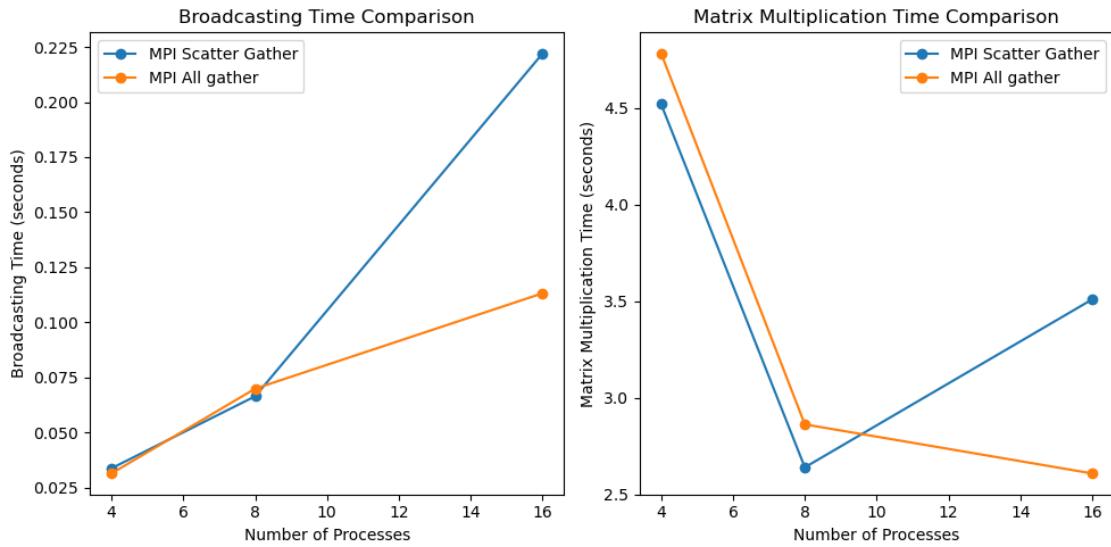
allgather_processes = [4, 8, 16]
allgather_broadcasting_time = [0.031401872634887695, 0.06987953186035156, 0.
    ↪11317896842956543]
allgather_multiplication_time = [4.781134605407715, 2.8613476753234863, 2.
    ↪607423782348633]

plt.figure(figsize=(10, 5))

plt.subplot(1, 2, 1)
plt.plot(scatter_gather_processes, scatter_gather_broadcasting_time, □
    ↪marker='o', label='MPI Scatter Gather')
plt.plot(allgather_processes, allgather_broadcasting_time, marker='o', □
    ↪label='MPI All gather')
plt.xlabel('Number of Processes')
plt.ylabel('Broadcasting Time (seconds)')
plt.title('Broadcasting Time Comparison')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(scatter_gather_processes, scatter_gather_multiplication_time, □
    ↪marker='o', label='MPI Scatter Gather')
plt.plot(allgather_processes, allgather_multiplication_time, marker='o', □
    ↪label='MPI All gather')
plt.xlabel('Number of Processes')
plt.ylabel('Matrix Multiplication Time (seconds)')
plt.title('Matrix Multiplication Time Comparison')
plt.legend()

plt.tight_layout()
plt.show()
```



Assignment 12

Derived data Type

1. Write about derived data types used in MPI programming.
 - Derived data types in MPI (Message Passing Interface) programming are custom data structures formed by combining basic MPI data types such as integers, floating-point numbers, and characters. These custom data types enable the efficient transmission of complex data structures between MPI processes. Structures, arrays of structures, and arrays of arrays are examples of common derived data types.
2. Steps to create and use derived data types.
 - The Procedure to Create and Use Derived Data Types in MPI:
 - Define the data type's structure using the `MPI_Type_struct` function.
 - Use the `MPI_Type_create_struct` function to specify the displacement and size of each element in the structure.
 - Use `MPI_Type_commit` to commit the newly created data type.
 - Use the derived data type for MPI communication functions like `MPI_Send` and `MPI_Recv`.
3. Write its uses.
 - Applications of derived data types in MPI:
 - Efficient transmission of complex data structures: Derived data types enable the transmission of structured data, such as arrays or nested structures, via a single MPI communication call.
 - Improved performance: By using custom data types tailored to the application's data layout, MPI communication can be made more efficient.
 - Simplified communication code: Derived data types abstract the complexities of data transmission, making code cleaner and easier to maintain.
4. Implement communication of derived data using one suitable example.

```
[1]: from mpi4py import MPI
import numpy as np
```

```
[2]: comm = MPI.COMM_WORLD
rank = comm.Get_rank()
size = comm.Get_size()
```

```
[23]: class Particle:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    send_data = None
    if rank == 0:
        send_data = Particle(6, 46)

    recv_data = comm.scatter([send_data] * size, root=0)

    print(f"Process {rank} received data: x={recv_data.x}, y={recv_data.y}")

MPI.Finalize()
```

Process 0 received data: x=6, y=46

```
[24]: !mpiexec -n 8 python as-12.py
```

Process 5 received data: x=6, y=46
 Process 4 received data: x=6, y=46
 Process 1 received data: x=6, y=46
 Process 0 received data: x=6, y=46
 Process 2 received data: x=6, y=46
 Process 3 received data: x=6, y=46
 Process 6 received data: x=6, y=46
 Process 7 received data: x=6, y=46

```
[25]: !mpiexec -n 16 python as-12.py
```

Process 2 received data: x=6, y=46
 Process 9 received data: x=6, y=46
 Process 1 received data: x=6, y=46
 Process 10 received data: x=6, y=46
 Process 7 received data: x=6, y=46
 Process 8 received data: x=6, y=46
 Process 3 received data: x=6, y=46
 Process 6 received data: x=6, y=46
 Process 13 received data: x=6, y=46
 Process 11 received data: x=6, y=46
 Process 5 received data: x=6, y=46
 Process 12 received data: x=6, y=46
 Process 14 received data: x=6, y=46
 Process 4 received data: x=6, y=46
 Process 15 received data: x=6, y=46
 Process 0 received data: x=6, y=46

Assignment 13

Device Query

❖ Try the below mentioned commands, explain their task in one line and paste the output for each of them.

- All commands run under the WSL (windows subsystem for linux).

➤ **lshw (List Hardware)**

This command list all the hardware details of system. By default it give all of the information to get an specific information command can be filter as below :

```
sudo lshw -C [option]
```

Option	usage
network	Gets the details of the network hardware devices.
memory	Displays the details of RAM.
storage	Show the details of the storage.
system	show the details of the motherboard and other things.
multimedia	Details of the sound card of system.
display	Know more about what is powering the display output.
bridge	Displays info about the PCIe bridges.
bus	It will list down buses and their details.
CPU	List the processor details

Output :

```
kavan@Kavan:~$ lshw
WARNING: you should run this program as super-user.

kavan
      description: Computer
      width: 64 bits
      capabilities: smp vsyscall32
*-core
      description: Motherboard
      physical id: 0
*-memory
      description: System memory
      physical id: 0
      size: 8064MiB
*-cpu
      product: 13th Gen Intel(R) Core(TM) i7-13620H
      vendor: Intel Corp.
      physical id: 1
      bus info: cpu@0
      version: 6.186.2
      width: 64 bits
      capabilities: fpu fpu_exception wp vme de pse tsc msr pae mce cx8 apic sep mttr
      pge mca cmov pat pse36 clflush mmx fxsr sse sse2 ss ht syscall nx pdpe1gb rdtscp x86-
      64 constant_tsc rep_good nopl xtopology tsc_reliable nonstop_tsc cpuid pni pclmulqdq
      vmx ssse3 fma cx16 sse4_1 sse4_2 x2apic movbe popcnt tsc_deadline_timer aes xsave
      avx f16c rdrand hypervisor lahf_lm abm 3dnowprefetch ssbd ibrs ibpb stibp
      ibrs_enhanced tpr_shadow vnmi ept vpid ept_ad fsgsbase tsc_adjust bmi1 avx2 smep
      bmi2 erms invpcid rdseed adx smap clflushopt clwb sha_ni xsaveopt xsavec xgetbv1
      xsaves avx_vnni umip waitpkg gfni vaes vpclmulqdq rdpid movdiri movdir64b fsrm
      serialize flush_l1d arch_capabilities

      configuration: microcode=4294967295
*-generic
```

description: System peripheral
product: Virtio file system
vendor: Red Hat, Inc.
physical id: 2
bus info: pci@4257:00:00.0
version: 01
width: 64 bits
clock: 33MHz
capabilities: bus_master cap_list
configuration: driver=virtio-pci latency=64
resources: iomemory:e0-df iomemory:e0-df iomemory:c0-bf irq:0
memory:e00000000-e00000fff memory:e00001000-e00001fff memory:c00000000-
dfffffff
*-virtio1 UNCLAIMED
description: Virtual I/O device
physical id: 0
bus info: virtio@1
configuration: driver=virtiofs
*-display:0
description: 3D controller
product: Microsoft Corporation
vendor: Microsoft Corporation
physical id: 3
bus info: pci@8383:00:00.0
version: 00
width: 32 bits
clock: 33MHz
capabilities: bus_master cap_list
configuration: driver=dxgkrnl latency=0
resources: irq:0

*-scsi

 description: SCSI storage controller

 product: Virtio console

 vendor: Red Hat, Inc.

 physical id: 4

 bus info: pci@8f33:00:00.0

 version: 01

 width: 64 bits

 clock: 33MHz

 capabilities: scsi bus_master cap_list

 configuration: driver=virtio-pci latency=64

 resources: iomemory:90-8f iomemory:90-8f iomemory:90-8f irq:0
 memory:9ffe00000-9ffe00fff memory:9ffe01000-9ffe01fff memory:9ffe02000-9ffe02fff

*-virtio0 UNCLAIMED

 description: Virtual I/O device

 physical id: 0

 bus info: virtio@0

 configuration: driver=virtio_console

*-display:1

 description: 3D controller

 product: Microsoft Corporation

 vendor: Microsoft Corporation

 physical id: 5

 bus info: pci@abca:00:00.0

 version: 00

 width: 32 bits

 clock: 33MHz

 capabilities: bus_master cap_list

 configuration: driver=dxgkrnl latency=0

 resources: irq:0

```
*-pnp00:00
    product: PnP device PNP0b00
    physical id: 6
    capabilities: pnp
    configuration: driver=rtc_cmos

*-network
    description: Ethernet interface
    physical id: 1
    logical name: eth0
    serial: 00:15:5d:ba:ac:bb
    size: 10Gbit/s
    capabilities: ethernet physical
    configuration: autonegotiation=off broadcast=yes driver=hv_netvsc
driverversion=5.15.146.1-microsoft-standard-W duplex=full firmware=N/A
ip=172.19.231.200 link=yes multicast=yes speed=10Gbit/s

WARNING: output may be incomplete or inaccurate, you should run this program as
super-user.
```

➤ **lsusb (List USB Devices)**

The lsusb command is a utility in Linux that allows users to list the USB (Universal Serial Bus) devices connected to the system.

Output :

```
kavan@Kavan:~$ lsusb
Bus 002 Device 001: ID 1d6b:0003 Linux Foundation 3.0 root hub
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
```

➤ **lspci (List PCI Devices)**

The lspci (list PCI) command is a Linux utility that displays detailed information about all PCI buses and devices in the system. It is based on the libpci library, which provides access to the PCI configuration space on a variety of operating systems.

Output :

```
kavan@Kavan:~$ lspci  
4257:00:00.0 System peripheral: Red Hat, Inc. Virtio file system (rev 01)  
8383:00:00.0 3D controller: Microsoft Corporation Device 008e  
8f33:00:00.0 SCSI storage controller: Red Hat, Inc. Virtio console (rev 01)  
abca:00:00.0 3D controller: Microsoft Corporation Device 008e
```

➤ **lsblk (List Block Devices)**

The 'lsblk' stands for 'list block devices', The lsblk command is a Linux command-line utility that lists information about all block devices on the system. This includes hard disk drives (HDDs), solid-state drives (SSDs), optical drives, and other storage devices.

Output :

```
kavan@Kavan:~$ lsblk  
NAME MAJ:MIN RM SIZE RO TYPE MOUNTPOINTS  
sda 8:0 0 388.5M 1 disk  
sdb 8:16 0 2G 0 disk [SWAP]  
sdc 8:32 0 1T 0 disk /snap  
    /mnt/wslg/distro  
    /
```

➤ **lscpu (List CPU)**

The lscpu command displays a variety of information about the CPU architecture, including The number of CPUs, The number of threads, The number of cores, The number of sockets, The cache details, The CPU architecture, The CPU vendor, The CPU model, The CPU frequency, The CPU flags.

Output :

```
kavan@Kavan:~$ lscpu  
Architecture:      x86_64  
CPU op-mode(s):   32-bit, 64-bit
```

Address sizes: 39 bits physical, 48 bits virtual

Byte Order: Little Endian

CPU(s): 16

On-line CPU(s) list: 0-15

Vendor ID: GenuineIntel

Model name: 13th Gen Intel(R) Core(TM) i7-13620H

CPU family: 6

Model: 186

Thread(s) per core: 2

Core(s) per socket: 8

Socket(s): 1

Stepping: 2

BogoMIPS: 5836.79

Flags: fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat
pse36 clflush mmx fxsr sse sse2 ss ht syscall nx pdpe1gb rdtscp lm constant_tsc
rep_good

nopl xtopology tsc_reliable nonstop_tsc cpuid pnpi pclmulqdq vmx ssse3
fma cx16 sse4_1 sse4_2 x2apic movbe popcnt tsc_deadline_timer aes xsave avx f16c rd

rand hypervisor lahf_lm abm 3dnowprefetch ssbd ibrs ibpb stibp
ibrs_enhanced tpr_shadow vnmi ept vpid ept_ad fsgsbase tsc_adjust bmi1 avx2 smep
bmi2 erms

invpcid rdseed adx smap clflushopt clwb sha_ni xsaveopt xsavec xgetbv1
xsaves avx_vnni umip waitpkg gfni vaes vpclmulqdq rdpid movdir64b fsrm seri
alize flush_l1d arch_capabilities

Virtualization features:

Virtualization: VT-x

Hypervisor vendor: Microsoft

Virtualization type: full

Caches (sum of all):

L1d: 384 KiB (8 instances)

L1i: 256 KiB (8 instances)

L2: 10 MiB (8 instances)

L3: 24 MiB (1 instance)

Vulnerabilities:

Gather data sampling: Not affected

Itlb multihit: Not affected

L1tf: Not affected

Mds: Not affected

Meltdown: Not affected

Mmio stale data: Not affected

Retbleed: Mitigation; Enhanced IBRS

Spec rstack overflow: Not affected

Spec store bypass: Mitigation; Speculative Store Bypass disabled via prctl and seccomp

Spectre v1: Mitigation; usercopy/swaps barriers and __user pointer sanitization

Spectre v2: Mitigation; Enhanced IBRS, IBPB conditional, RSB filling, PBRSB-eIBRS SW sequence

Srbds: Not affected

Tsx async abort: Not affected

➤ **df (Disk Free)**

The df command in Linux is used to display the amount of disk space available on the filesystem. The FileSystem parameter specifies the name of the device on which the file system resides, the directory on which the file system is mounted, or the relative path name of a file system.

Output :

kavan@Kavan:~\$ df

Filesystem	1K-blocks	Used	Available	Use%	Mounted on
none	3997904	4	3997900	1%	/mnt/wsl
none	395990012136101968	259888044	35%	/usr/lib/wsl/drivers	
none	3997904	0	3997904	0%	/usr/lib/modules

```

none      3997904    0  3997904  0% /usr/lib/modules/5.15.146.1-microsoft-
standard-WSL2

/dev/sdc   1055762868 1902524 1000156872 1% /

none      3997904    84  3997820  1% /mnt/wslg

none      3997904    0  3997904  0% /usr/lib/wsl/lib

rootfs    3994648   1884  3992764  1% /init

none      3997904    808  3997096  1% /run

none      3997904    0  3997904  0% /run/lock

none      3997904    0  3997904  0% /run/shm

tmpfs     4096     0   4096  0% /sys/fs/cgroup

none      3997904    76  3997828  1% /mnt/wslg/versions.txt

none      3997904    76  3997828  1% /mnt/wslg/doc

C:\      395990012 136101968 259888044 35% /mnt/c

D:\      601881596 406166716 195714880 68% /mnt/d

G:\      15728640 12185968 3542672 78% /mnt/g

H:\      15728640 458840 15269800 3% /mnt/h

I:\      15728640 786924 14941716 6% /mnt/i

J:\      395990012 149096372 246893640 38% /mnt/j

snapfuse   128    128    0 100% /snap/bare/5

snapfuse   75776   75776   0 100% /snap/core22/864

snapfuse   93952   93952   0 100% /snap/gtk-common-themes/1535

snapfuse   41856   41856   0 100% /snap/snapd/20290

snapfuse   134272  134272   0 100% /snap/ubuntu-desktop-installer/1276

snapfuse   134912  134912   0 100% /snap/ubuntu-desktop-installer/1286

```

➤ ip a (IP Address)

The ip a command in Linux is used to display the network interface addresses and routing table of the system. This will display a list of all the network interfaces on the system, along with their IP addresses, subnet masks, and default gateways.

Output :

```
kavan@Kavan:~$ ip a  
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group  
default qlen 1000  
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00  
    inet 127.0.0.1/8 scope host lo  
        valid_lft forever preferred_lft forever  
    inet6 ::1/128 scope host  
        valid_lft forever preferred_lft forever  
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group  
default qlen 1000  
    link/ether 00:15:5d:ba:ac:bb brd ff:ff:ff:ff:ff:ff  
    inet 172.19.231.200/20 brd 172.19.239.255 scope global eth0  
        valid_lft forever preferred_lft forever  
    inet6 fe80::215:5dff:feba:acbb/64 scope link  
        valid_lft forever preferred_lft forever
```

➤ **top**

The top command in Linux is a very useful tool for monitoring system performance. It provides a real-time view of running processes, CPU and memory usage, and other system information. the list shows the process ID (PID), username, CPU usage, memory usage, and command name.

Output :

```

kavan@Kavan:~$ top
top - 20:14:07 up 41 min,  1 user,  load average: 0.02, 0.01, 0.00
Tasks: 33 total,   1 running, 32 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.1 us,  0.1 sy,  0.0 ni, 99.8 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
MiB Mem :  7808.4 total,  6951.5 free,   507.2 used,   349.7 buff/cache
MiB Swap:  2048.0 total,   2048.0 free,      0.0 used.  7071.6 avail Mem

          PID USER      PR  NI    VIRT    RES   SHR S %CPU %MEM TIME+ COMMAND
        1 root      20   0 165860 11216 8244 S  1.0  0.1 0:21.48 systemd
      426 root      20   0 44224 38752 10428 S  0.7  0.5 0:12.13 python3
        2 root      20   0  2280  1300 1188 S  0.0  0.0 0:00.00 init-systemd(Ub
       11 root      20   0  2280     4  0 S  0.0  0.0 0:00.00 init
      40 root      19  -1 47732 15248 14248 S  0.0  0.2 0:00.09 systemd-journal
      60 root      20   0 22092  6040 4544 S  0.0  0.1 0:00.21 systemd-udevd
      71 root      20   0  4848  1808 1340 S  0.0  0.0 0:00.91 snapfuse
      72 root      20   0  4492  180   36 S  0.0  0.0 0:00.00 snapfuse
      75 root      20   0  4624  160     8 S  0.0  0.0 0:00.00 snapfuse
      80 root      20   0  4492  168   20 S  0.0  0.0 0:00.00 snapfuse
      83 root      20   0  4764  1764 1228 S  0.0  0.0 0:02.56 snapfuse
      87 root      20   0  4492  160     8 S  0.0  0.0 0:00.00 snapfuse
      89 root      20   0  4760  1780 1228 S  0.0  0.0 0:01.10 snapfuse
     96 systemd+  20   0 25532 12780 8492 S  0.0  0.2 0:00.09 systemd-resolve
    137 root      20   0  4304  2656 2416 S  0.0  0.0 0:00.00 cron
   143 message+  20   0  8584  4616 4072 S  0.0  0.1 0:00.02 dbus-daemon
   158 root      20   0 30096 19212 10432 S  0.0  0.2 0:00.06 networkd-dispat
   160 syslog    20   0 222400  7084 4272 S  0.0  0.1 0:00.02 rsyslogd
   162 root      20   0 2058580 48152 19360 S  0.0  0.6 0:00.73 snapd
   163 root      20   0  15324  7456 6512 S  0.0  0.1 0:00.07 systemd-logind
   217 root      20   0  4780  3352 3108 S  0.0  0.0 0:00.06 subiquity-serve
   225 root      20   0  3236  1084  996 S  0.0  0.0 0:00.00 agetty
   228 root      20   0  3192  1168 1080 S  0.0  0.0 0:00.00 agetty
   239 root      20   0 107224 21372 13268 S  0.0  0.3 0:00.04 unattended-upgr
   364 root      20   0 934936 87296 24796 S  0.0  1.1 0:04.26 python3.10
   369 root      20   0  2284   112     0 S  0.0  0.0 0:00.00 SessionLeader
   370 root      20   0  2300   116     0 S  0.0  0.0 0:00.00 Relay(371)
   371 kavan     20   0  6208  5064 3344 S  0.0  0.1 0:00.02 bash
   372 root      20   0  7516  4864 3948 S  0.0  0.1 0:00.00 login
   408 kavan     20   0 16916  9024 7564 S  0.0  0.1 0:00.03 systemd
   409 kavan     20   0 168912  3424   12 S  0.0  0.0 0:00.00 (sd-pam)
   414 kavan     20   0  6120  4968 3384 S  0.0  0.1 0:00.01 bash
 10182 kavan    20   0  7940   3700 3104 R  0.0  0.0 0:00.02 top

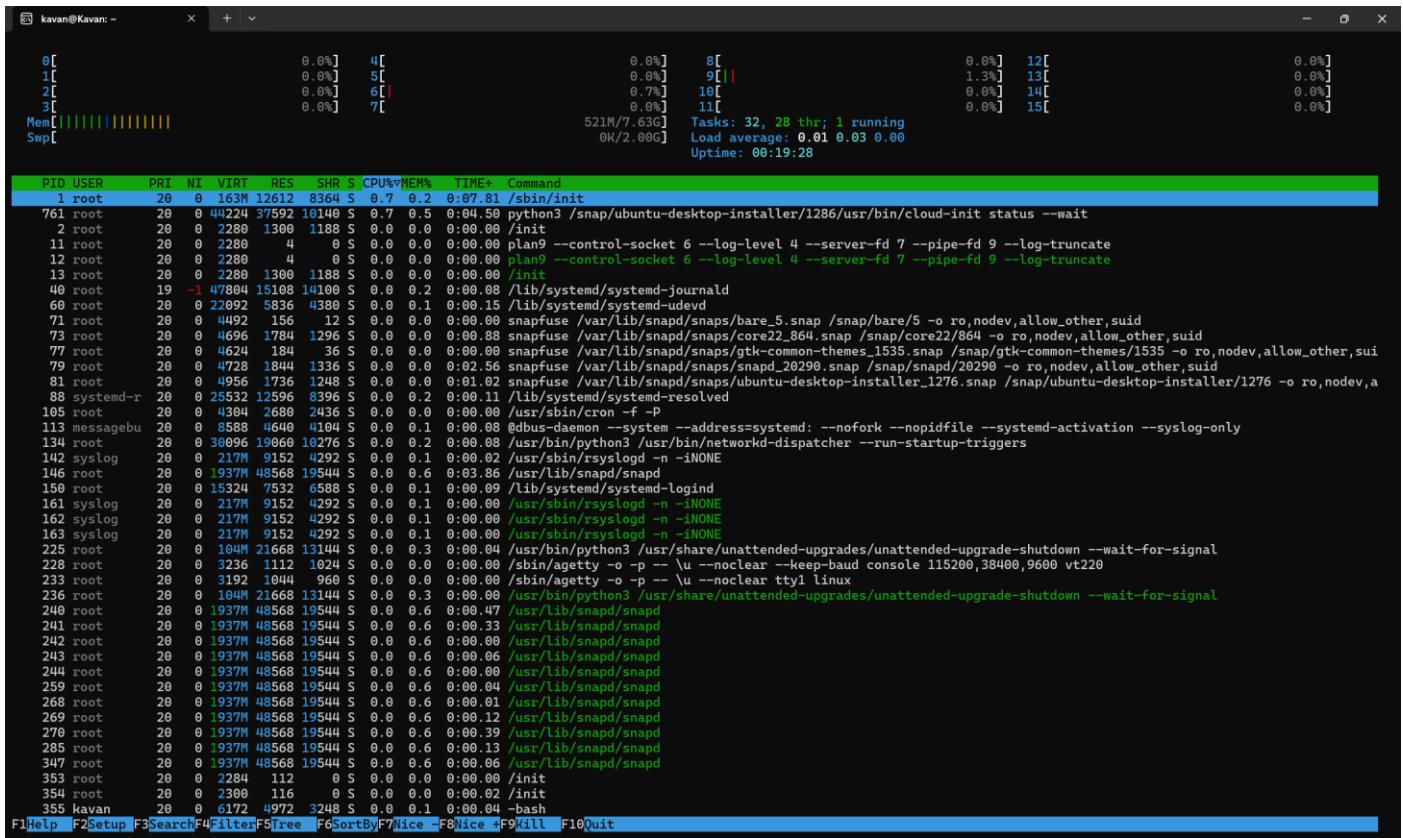
```

➤ htop

The htop command is a Linux system monitor that provides real-time information about the system's CPU usage, memory, and running processes. It is similar to the top command, but it offers a number of advantages, including:

A more user-friendly interface, The ability to sort processes by different criteria, The ability to kill processes directly from the htop interface, The ability to view detailed information about individual processes

Output :



➤ nvidia-smi

The `nvidia-smi` command is a Linux command-line utility that provides monitoring and management capabilities for NVIDIA GPUs. This displays the status of an NVIDIA GPU, such as its temperature, utilization, and memory usage.

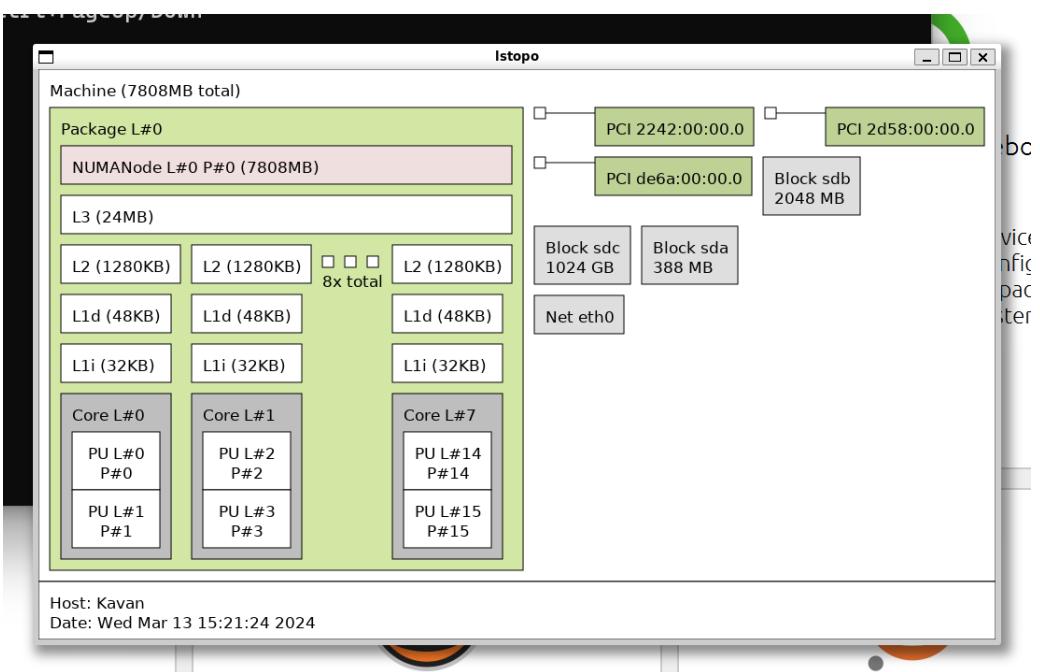
Output :

```
kavan@Kavan:~$ nvidia-smi
Wed Mar 13 20:32:45 2024
+-----+
| NVIDIA-SMI 550.60.01      Driver Version: 551.76      CUDA Version: 12.4 |
+-----+
| GPU  Name     Persistence-M | Bus-Id     Disp.A  Volatile Uncorr. ECC | | | | |
| Fan  Temp     Perf          Pwr:Usage/Cap | Memory-Usage | GPU-Util Compute M. |
|          |             |             |           |           |          MIG M. |
+-----+
|   0  NVIDIA GeForce RTX 4060 ...  On | 00000000:01:00.0  On |                N/A | | |
| N/A   69C   P0    94W / 120W | 5205MiB / 8188MiB | 99%       Default |
|          |             |           |           |          N/A |
+-----+
+-----+
| Processes:                               GPU Memory |
| GPU  GI  CI      PID  Type  Process name        Usage  |
| ID   ID          ID          |
+-----+
| No running processes found            |
+-----+
```

➤ **lstopo**

The *lstopo* command is a Linux command that displays the topology of a system. This means that it shows how the different hardware components of the system are connected to each other.

Output :



➤ **numactl**

The *numactl* command is a utility that allows users to control the NUMA (Non-Uniform Memory Access) policy for processes or shared memory. It can be used to set the memory and processor affinity of a process, as well as to set a persistent policy for shared memory segments or files.

Output :

```
kavan@Kavan:~$ numactl
```

```
usage: numactl [--all | -a] [--interleave= | -i <nodes>] [--preferred= | -p <node>]
                [--physcpubind= | -C <cpus>] [--cpunodebind= | -N <nodes>]
                [--membind= | -m <nodes>] [--localalloc | -l] command args ...
numactl [--show | -s]
numactl [--hardware | -H]
```

```
numactl [--length | -l <length>] [--offset | -o <offset>] [--shmmode | -M <shmmode>]
[--strict | -t]
[--shmid | -I <id>] --shm | -S <shmkeyfile>
[--shmid | -I <id>] --file | -f <tmpfsfile>
[--huge | -u] [--touch | -T]
memory policy | --dump | -d | --dump-nodes | -D
```

memory policy is --interleave | -i, --preferred | -p, --membind | -m, --localalloc | -l
<nodes> is a comma delimited list of node numbers or A-B ranges or all.

Instead of a number a node can also be:

netdev:DEV the node connected to network device DEV
file:PATH the node the block device of path is connected to
ip:HOST the node of the network device host routes through
block:PATH the node of block device path
pci:[seg:]bus:dev[:func] The node of a PCI device

<cpus> is a comma delimited list of cpu numbers or A-B ranges or all
all ranges can be inverted with !

all numbers and ranges can be made cpuset-relative with +
the old --cpubind argument is deprecated.
use --cpunodebind or --physcpubind instead
<length> can have g (GB), m (MB) or k (KB) suffixes

➤ **sar**

The sar (System Activity Reporter) command is used for monitoring system performance in Linux. It can be used to collect, report, and save system activity information, such as CPU usage, memory usage, I/O activity, and network traffic.

Output :

To generate a report of CPU usage every second for 5 seconds, used the following command:

sar -u 15

```
kavan@Kavan:~$ sar -u 1 5
Linux 5.15.146.1-microsoft-standard-WSL2 (Kavan)          03/13/24      _x86_64_      (16 CPU)

22:16:09      CPU    %user    %nice   %system   %iowait   %steal   %idle
22:16:10      all     0.12     0.00     0.06     0.00     0.00     99.81
22:16:11      all     0.06     0.00     0.12     0.00     0.00     99.81
22:16:12      all     0.06     0.00     0.00     0.00     0.00     99.94
22:16:13      all     0.06     0.00     0.06     0.00     0.00     99.87
22:16:14      all     0.12     0.00     0.00     0.00     0.00     99.88
Average:      all     0.09     0.00     0.05     0.00     0.00     99.86
```

To generate a report of memory usage every minute for 5 minutes, used the following command:

sar -r 60 5

System Resource Utilization - 03/13/24											
CPU Usage				Memory Usage				Disk I/O			
Process		Utilization		Memory		Cache		Throughput		Latency	
User	System	Resident Set Size	Working Set Size	Total	Used	Free	Used	Reads/s	Writes/s	Avg. Latency	Max. Latency
kavan@Kavan:~\$	sar -r 10 5										
Linux 5.15.146.1-microsoft-standard-WSL2 (Kavan)				03/13/24				_x86_64_			
22:16:18	kbmemfree	kbavail	kbmemused	%memused	kbbuffers	kbcached	kbboxmit	%commit	kbaactive	kbinact	kbdirty
22:16:28	6701132	7181868	508012	6.35	15140	662072	889324	8.81	135180	792420	0
22:16:38	6701196	7181932	507944	6.35	15140	662072	889324	8.81	135180	792484	0
22:16:48	6701200	7181936	507936	6.35	15140	662072	889324	8.81	135180	792488	0
22:16:58	6700988	7181724	508140	6.36	15140	662072	889324	8.81	135180	792488	0
22:17:08	6700508	7181248	508604	6.36	15148	662072	889324	8.81	135180	792472	20
Average:	6701805	7181742	508127	6.35	15142	662072	889324	8.81	135180	792470	4

To generate a report of I/O activity every 1 seconds for 5 seconds, used the following command:

sar -d 5 30

```

kavan@Kavan:~$ sar -d 1 5
Linux 5.15.146.1-microsoft-standard-WSL2 (Kavan)           03/13/24      _x86_64_      (16 CPU)

22:18:34   DEV    tps   rkB/s   wkB/s   dkB/s   areq-sz   aqu-sz   await   %util
22:18:35   sda   0.00   0.00   0.00   0.00   0.00   0.00   0.00   0.00   0.00
22:18:35   sdb   0.00   0.00   0.00   0.00   0.00   0.00   0.00   0.00   0.00
22:18:35   sdc   0.00   0.00   0.00   0.00   0.00   0.00   0.00   0.00   0.00

22:18:35   DEV    tps   rkB/s   wkB/s   dkB/s   areq-sz   aqu-sz   await   %util
22:18:36   sda   0.00   0.00   0.00   0.00   0.00   0.00   0.00   0.00   0.00
22:18:36   sdb   0.00   0.00   0.00   0.00   0.00   0.00   0.00   0.00   0.00
22:18:36   sdc   0.00   0.00   0.00   0.00   0.00   0.00   0.00   0.00   0.00

22:18:36   DEV    tps   rkB/s   wkB/s   dkB/s   areq-sz   aqu-sz   await   %util
22:18:37   sda   0.00   0.00   0.00   0.00   0.00   0.00   0.00   0.00   0.00
22:18:37   sdb   0.00   0.00   0.00   0.00   0.00   0.00   0.00   0.00   0.00
22:18:37   sdc   0.00   0.00   0.00   0.00   0.00   0.00   0.00   0.00   0.00

22:18:37   DEV    tps   rkB/s   wkB/s   dkB/s   areq-sz   aqu-sz   await   %util
22:18:38   sda   0.00   0.00   0.00   0.00   0.00   0.00   0.00   0.00   0.00
22:18:38   sdb   0.00   0.00   0.00   0.00   0.00   0.00   0.00   0.00   0.00
22:18:38   sdc   0.00   0.00   0.00   0.00   0.00   0.00   0.00   0.00   0.00

22:18:38   DEV    tps   rkB/s   wkB/s   dkB/s   areq-sz   aqu-sz   await   %util
22:18:39   sda   0.00   0.00   0.00   0.00   0.00   0.00   0.00   0.00   0.00
22:18:39   sdb   0.00   0.00   0.00   0.00   0.00   0.00   0.00   0.00   0.00
22:18:39   sdc   0.00   0.00   0.00   0.00   0.00   0.00   0.00   0.00   0.00

Average:   DEV    tps   rkB/s   wkB/s   dkB/s   areq-sz   aqu-sz   await   %util
Average:   sda   0.00   0.00   0.00   0.00   0.00   0.00   0.00   0.00   0.00
Average:   sdb   0.00   0.00   0.00   0.00   0.00   0.00   0.00   0.00   0.00
Average:   sdc   0.00   0.00   0.00   0.00   0.00   0.00   0.00   0.00   0.00

```

Assignment 14



Image Filtering

Presented by :

Shikhar Trivedi (23MDS012)

Kavan Mistry (23MDS006)

Meetrajsinh chudasama (23MDS009)

1

What is Image filtering ?

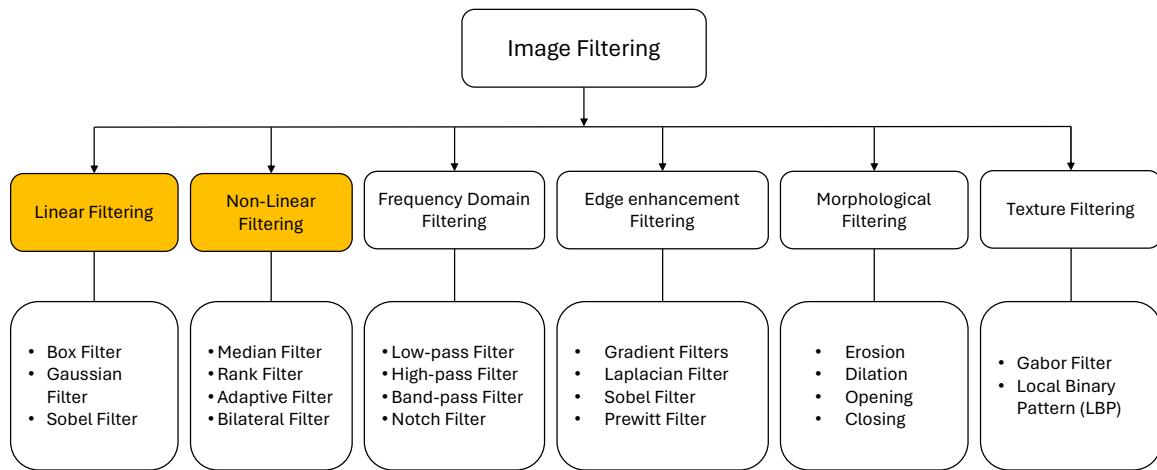
Image filter is the process of enhancing images by applying different filters , these enhancement vary from denoising , edge detection , smoothening or sharpening of image and plays a crucial role in any kind of image usage as such unnecessary elements like noises can hinder the image related processes , while certain noises can also be used to train the model for more robustness.

2

2

1

Types of Image filtering



3

3

Linear Filtering

Linear filter as the name says mostly deals with linear data or homogeneity (refers to constant change for example: a constant multiplied to matrix) , the output will also be of linear nature after convolution or multiplication with any constant. Linear filters are commonly used in image processing for various tasks such as smoothing, sharpening, noise reduction, and edge detection. Examples of linear filters include:

- Gaussian Filter
- Box Filter (Average Filter)
- Sobel Filter

4

4

2

Box/Average filter

Box filtering involves replacing each pixel of an image with the average in a box / filter for that bounding box. A Box Filter is a simple linear filter that applies a uniform filter kernel to the input image. It computes the average value of the pixel values within the kernel window and replaces the central pixel value with this average. This filter is effective for smoothing an image, reducing noise, and blurring edges. It is fast but can lead to loss of image details.

5

5

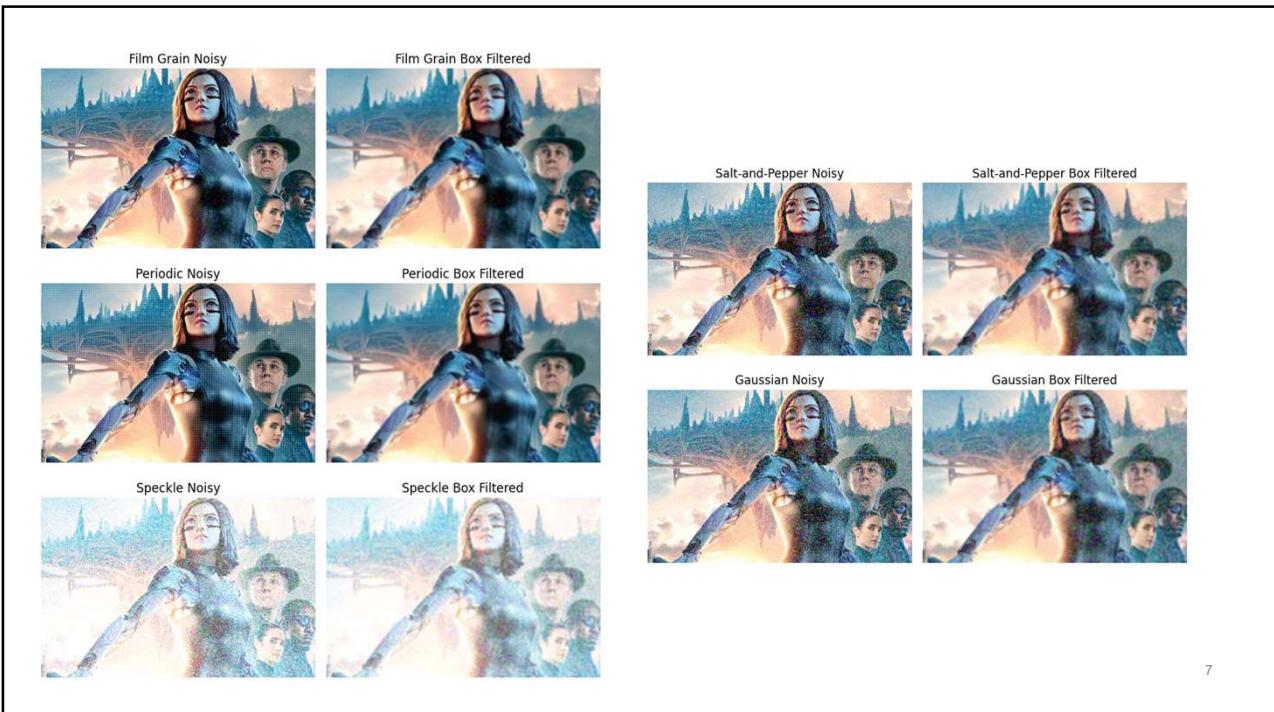
Box/Average filter output



6

6

3



7

7

Gaussian filter

The Gaussian Smoothing Operator performs a weighted average of surrounding pixels based on the Gaussian distribution. The Gaussian Filter applies a Gaussian kernel to the input image. It gives more weight to the central pixel and less weight to the surrounding pixels, following a Gaussian distribution. This filter effectively reduces noise while preserving edges better than the Box Filter. It is commonly used for image blurring which helps in increasing robustness of model and noise reduction/denoising , edge detection, feature extraction.

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

where, $G(x,y)$ is the value of the Gaussian kernel at coordinates (x,y)

σ (sigma) is the standard deviation of the Gaussian distribution, which determines the amount of blurring or smoothing applied to the image

(x,y) are the coordinates within the Gaussian kernel matrix

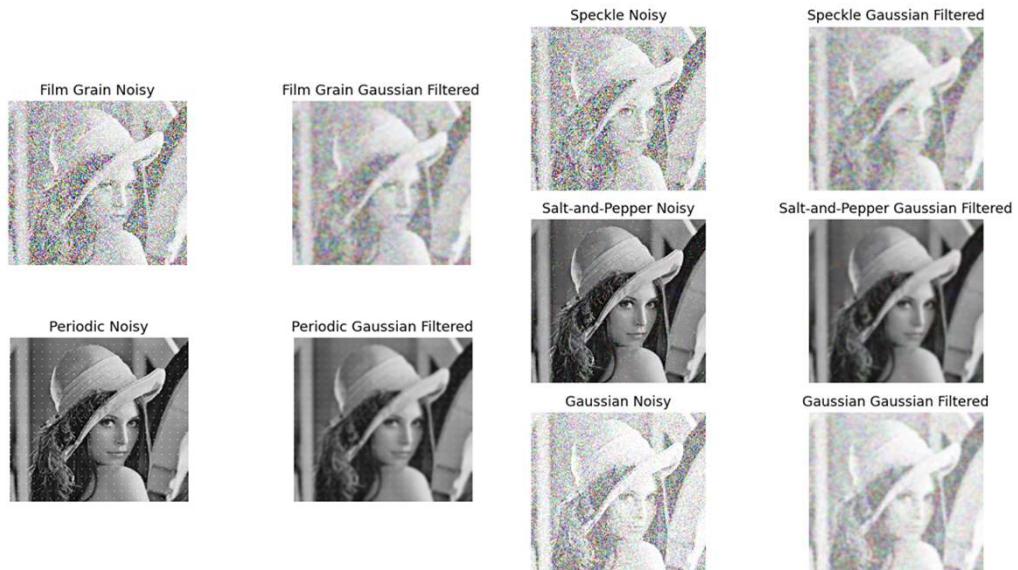
π (3.14) is used in the normalizing term

8

8

4

Gaussian filter output



9



10

10

5

Sobel filter

The Sobel filter is a gradient-based linear filter used in image processing for edge detection tasks. It computes the gradient magnitude of an image by convolving it with separate kernels for horizontal and vertical changes in pixel intensities. The Sobel filter emphasizes edges in both the horizontal and vertical directions, making it an effective tool for detecting edges and boundaries , image detection ,pre-processing(object detection).

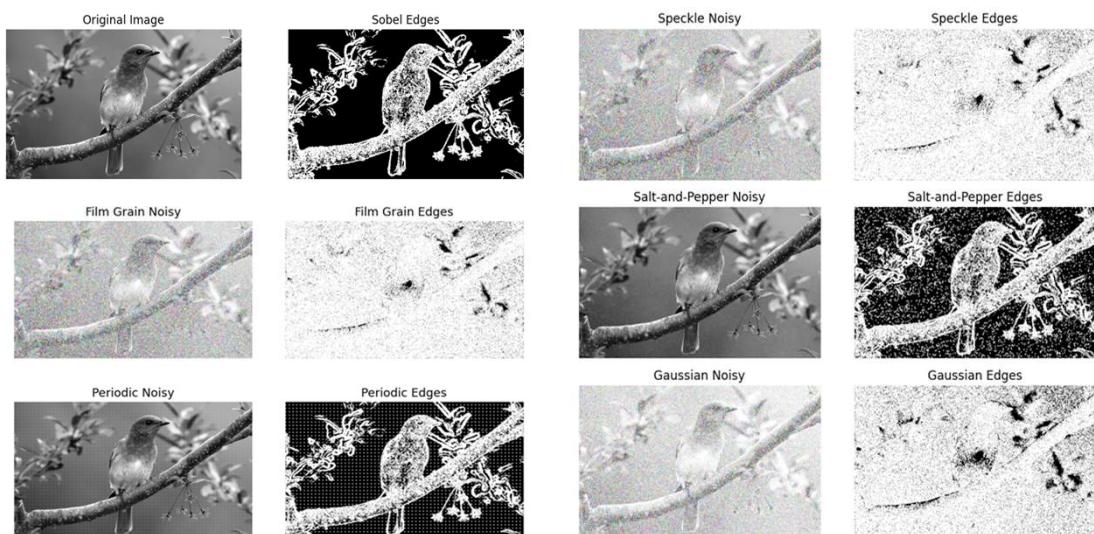
$$\Delta f = \text{mag}(\Delta f) = [G_x^2 + G_y^2]^{1/2}$$

$$\text{for } x - \text{direction : } \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}, \quad \text{for } y - \text{direction : } \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

11

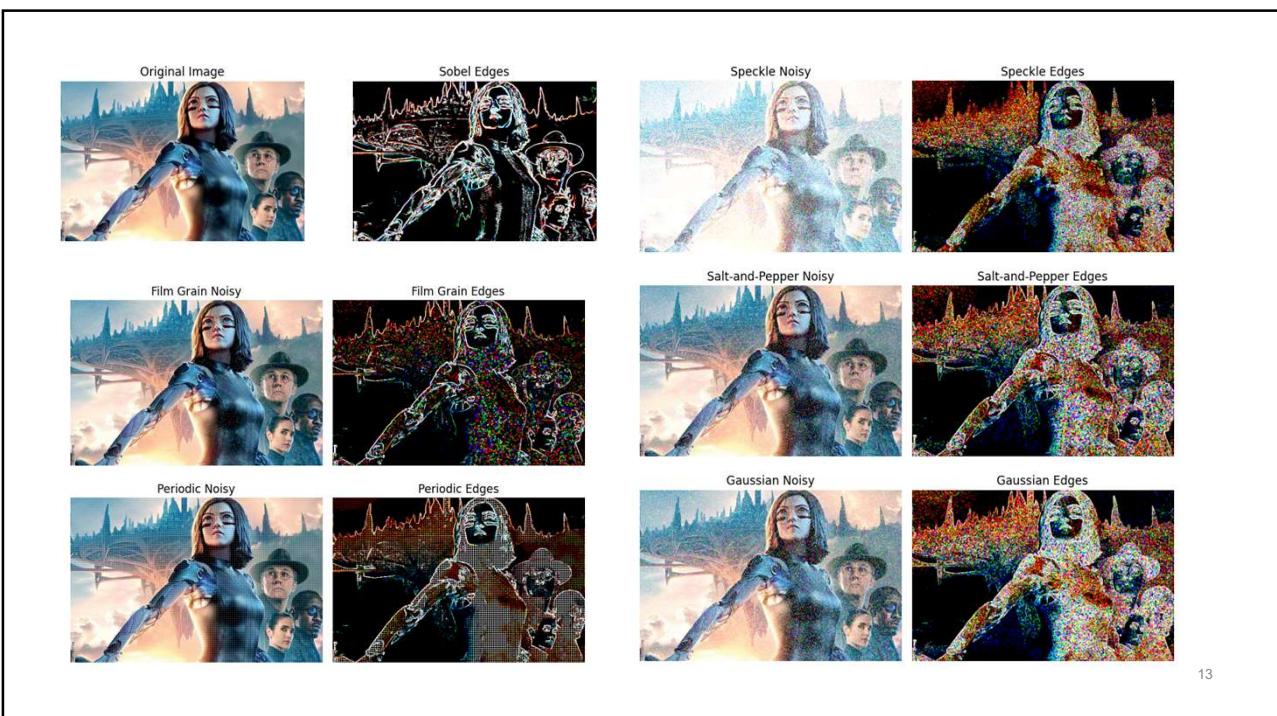
11

Sobel filter output



12

12



13

13

Non-Linear Filtering

Non-Linear filters do not follow the properties like homogeneity , linearity instead they use non-linear filters use more complex operations such as ranking, thresholding, and adaptive processing to modify and enhance images , non-linear filters are mostly used for details enhancement , noise reduction , edge preservation. Few examples of non-linear filters are:

- Median Filter
- Rank Filter
- Adaptive Filter
- Bilateral Filter

14

14

Median Filter

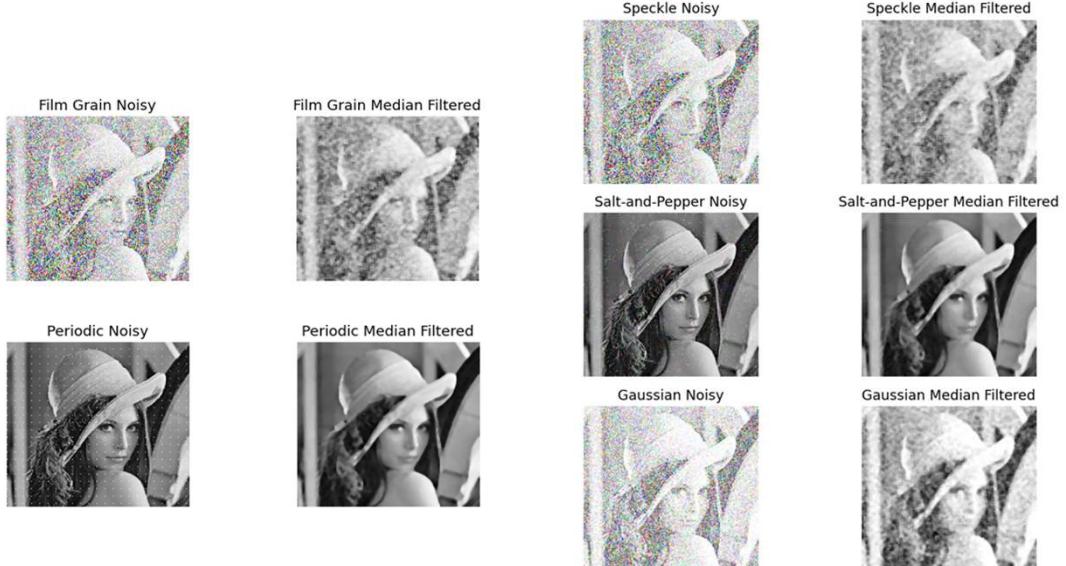
The Median Filter replaces each pixel's value with the median value within a specified neighborhood (kernel). It is effective for removing salt-and-pepper noise and preserving edges because it considers the pixel values rather than their intensities. This filter is computationally more expensive than linear filters but provides better noise reduction, irregularity removal, edge preservation and digital/signal processing.

$$\hat{f}(x,y) = \text{median}\{g(s,t)\} \quad , (s,t) \in S_{xy}$$

15

15

Median Filter output



16

16



17

17

Rank Filter

The Rank Filter is a generalized version of the Median Filter where the replacement value is determined by the rank (percentile) of the pixel values within the kernel window. It behaves like the Median Filter when the rank is set but can be customized to produce different effects based on the rank value resulting in details enhancement , edge preservation. Consists of max and min filters which together reduce noise.

$$\hat{f}(x,y) = \frac{1}{2} \left(\max_{(s,t) \in S_{xy}} \{g(s,t)\} + \min_{(s,t) \in S_{xy}} \{g(s,t)\} \right)$$

where, $\hat{f}(x,y)$ - This represents the output value of the rank filter at coordinates (x,y)

S_{xy} – This represents the set of all possible neighborhoods around the pixel located at (x,y)

(s,t) – These variables represent the coordinates within the neighborhood S_{xy}

$g(s,t)$ – This represents the intensity (or value) of the pixel at coordinates (s,t) within the neighborhood S_{xy}

$\max_{(s,t) \in S_{xy}} \{g(s,t)\}$ – calculates the maximum intensity value within the neighborhood S_{xy}

$\min_{(s,t) \in S_{xy}} \{g(s,t)\}$ – calculates the minimum intensity value within the neighborhood S_{xy}

18

18

Rank Filter output



19

19



20

20

10

Adaptive Filter

The Adaptive Filter, particularly Mean Shift Filtering, is a non-linear filter that iteratively shifts the pixel values towards the mode of the pixel distribution in the neighborhood. It effectively removes noise and preserves edges while reducing the computational cost compared to some other non-linear filters. Mean Shift Filtering is commonly used for image segmentation, texture analysis, and denoising tasks.

$$\hat{f}(x, y) = g(x, y) - \left(\frac{\sigma_\eta^2}{\sigma_L^2} \right) [g(x, y) - m_L]$$

where , σ_L^2 – Local variance of the local region

m_L – Local mean

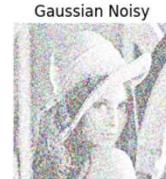
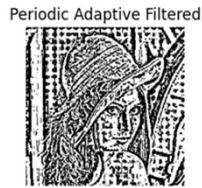
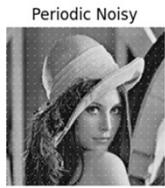
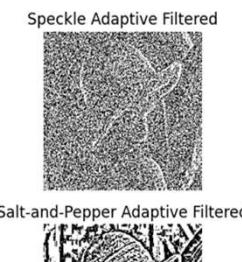
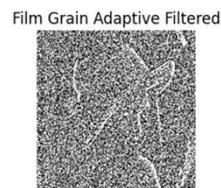
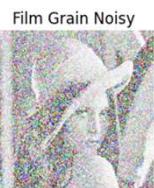
σ_η^2 – Variance of overall noise

$g(x, y)$ – Pixel value at the position (x, y)

21

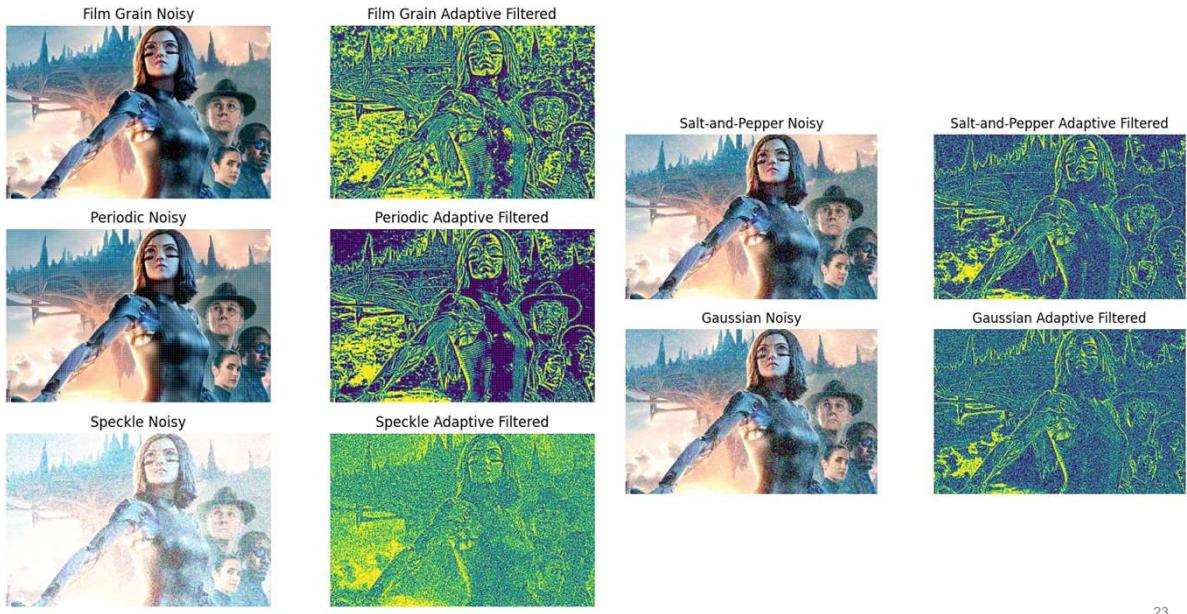
21

Adaptive Filter output



22

22



23

23

Bilateral Filter

The Bilateral Filter is a non-linear filter that considers both the spatial and intensity differences when filtering an image. It preserves edges while smoothing the image by applying a weighted average based on both the spatial and intensity differences between pixels. This filter is useful for noise reduction while preserving important image features, making it suitable for various tasks like HDR imaging, denoising, and edge-preserving smoothing.

$$BF[I]_p = \frac{1}{w_p} \sum_{q \in S} G_{\sigma_s}(|p - q|) G_{\sigma_r}(|I_p - I_q|) I_q$$

where, I represents the input image

p and q are pixel coordinates

S is the spatial domain, typically a window around p

w_p is a normalization factor

$\sum_{q \in S} G_{\sigma_s}(|p - q|)$ is the spatial Gaussian kernel, determined by the spatial distance $|p - q|$ and parameter σ_s controlling spatial smoothness

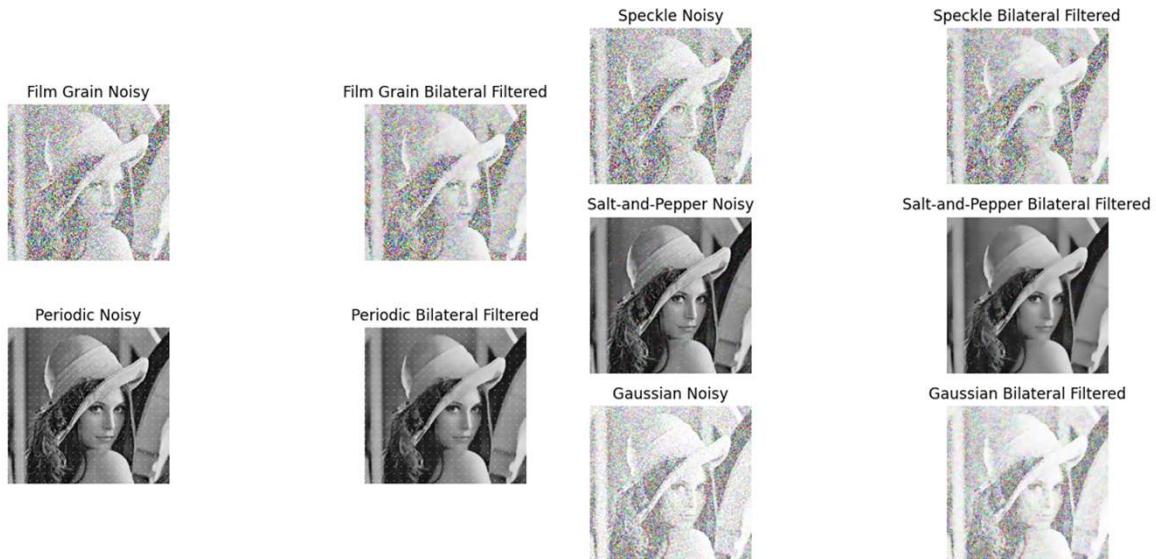
$G_{\sigma_r}(|I_p - I_q|)$ is the range Gaussian kernel, determined by the intensity difference $|I_p - I_q|$ and parameter σ_r controlling intensity similarity

24

24

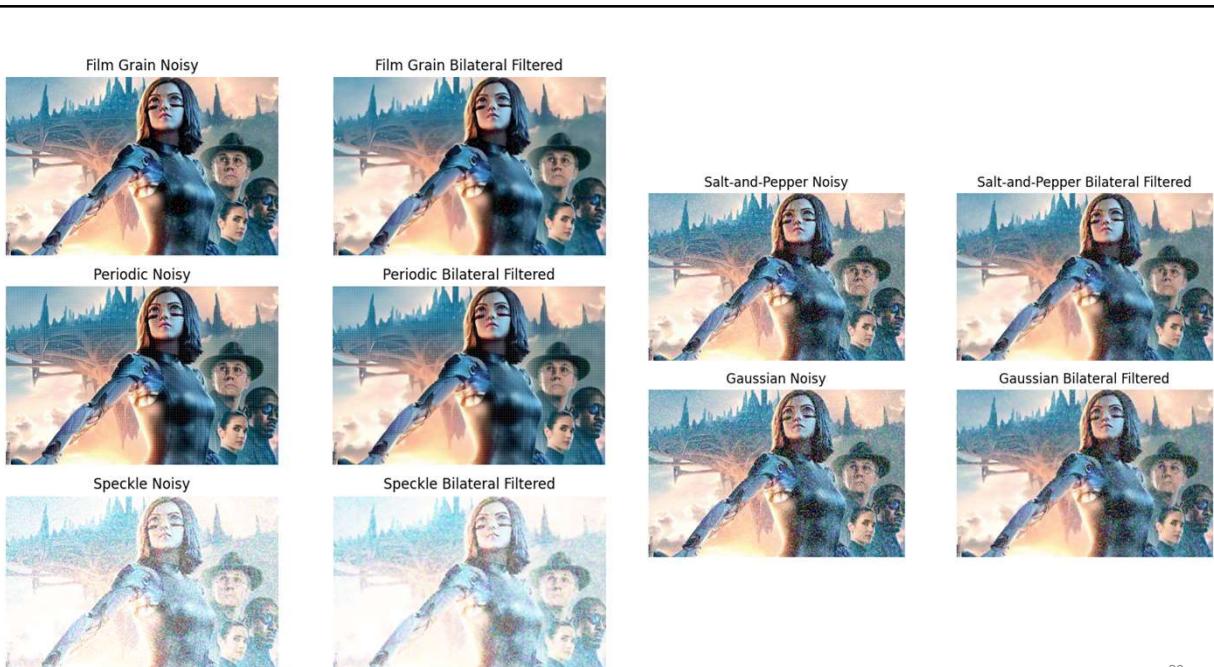
12

Bilateral Filter output



25

25



26

26

13



27

27

Assignment 15

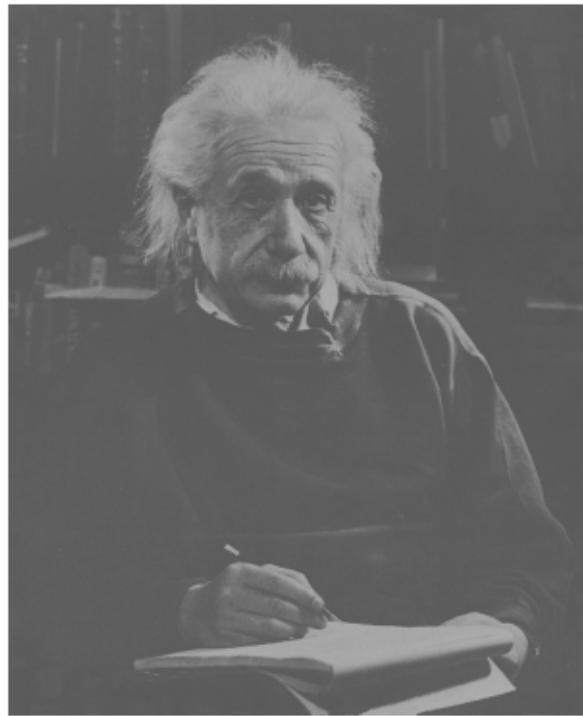
1 Image Filtering

```
[1]: import cv2
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image, ImageFilter
# from google.colab.patches import cv2_imshow
from skimage.filters import threshold_local
from matplotlib.gridspec import GridSpec
```

```
D:\anaconda\Lib\site-packages\paramiko\transport.py:219:
CryptographyDeprecationWarning: Blowfish has been deprecated
  "class": algorithms.Blowfish,
```

```
[2]: image_path = "I:\\My\\
Drive\\Image_filtering\\images\\images\\Fig0354(a)(einstein_orig).tif"
original_image = cv2.imread(image_path)
```

```
[3]: plt.imshow(original_image)
plt.axis('off')  # Hide axis
plt.show()
```



1.1 Image nosing

```
[4]: if original_image is None:
    print("Error: Unable to load the image.")
else:
    def generate_film_grain(image):
        film_grain_noise = np.random.normal(loc=0, scale=20, size=image.shape).
        ↪astype(np.uint8)
        return cv2.add(image, film_grain_noise)

    def generate_periodic(image):
        periodic_noise = np.zeros(image.shape, dtype=np.uint8)
        periodic_noise[::10, ::10, :] = 255
        return cv2.add(image, periodic_noise)

    def generate_speckle(image):
        speckle_noise = np.random.normal(loc=0, scale=0.1, size=image.shape) * ↪
        ↪255
        return cv2.add(image, speckle_noise.astype(np.uint8))

    def generate_salt_and_pepper(image):
        salt_pepper_noise = np.random.choice([0, 255], size=image.shape[:2] + ↪
        ↪(image.shape[2],), p=[0.99, 0.01]).astype(np.uint8)
```

```

    return cv2.add(image, salt_pepper_noise)

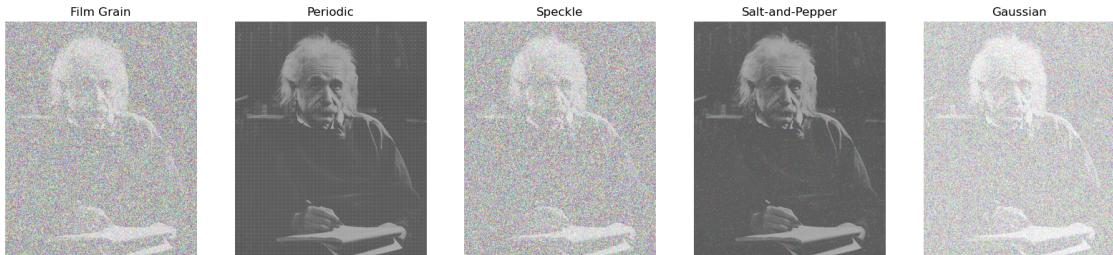
def generate_gaussian(image):
    gaussian_noise = np.random.normal(loc=0, scale=100, size=image.shape).
    ↪astype(np.uint8)
    return cv2.add(image, gaussian_noise)

noise_generators = {
    'Film Grain': generate_film_grain,
    'Periodic': generate_periodic,
    'Speckle': generate_speckle,
    'Salt-and-Pepper': generate_salt_and_pepper,
    'Gaussian': generate_gaussian
}

noisy_images = {noise_type: noise_generator(original_image) for noise_type, ↪
noise_generator in noise_generators.items()}

```

```
[5]: plt.figure(figsize=(20, 10))
for i, (noise_type, noisy_image) in enumerate(noisy_images.items(), start=1):
    plt.subplot(1, len(noisy_images), i)
    plt.imshow(cv2.cvtColor(noisy_image, cv2.COLOR_BGR2RGB))
    plt.title(noise_type)
    plt.axis('off')
plt.show()
```



1.2 Linear Filtering

1.2.1 Box Filter

```
[6]: def apply_box_filter(image):
    return cv2.boxFilter(image, -1, (5, 5))

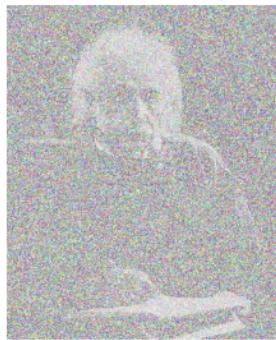
plt.figure(figsize=(8, 15))

for noise_type, noisy_image in noisy_images.items():

```

```
plt.subplot(len(noisy_images), 2, 2*(list(noisy_images.keys()).  
index(noise_type)) + 1)  
plt.imshow(cv2.cvtColor(noisy_image, cv2.COLOR_BGR2RGB))  
plt.title(noise_type + ' Noisy', fontsize=12)  
plt.axis('off')  
  
box_filtered = apply_box_filter(noisy_image)  
  
plt.subplot(len(noisy_images), 2, 2*(list(noisy_images.keys()).  
index(noise_type)) + 2)  
plt.imshow(cv2.cvtColor(box_filtered, cv2.COLOR_BGR2RGB))  
plt.title(noise_type + ' Box Filtered', fontsize=12)  
plt.axis('off')  
  
plt.tight_layout()  
plt.savefig('box_filter_ppt.png', bbox_inches='tight')  
plt.show()
```

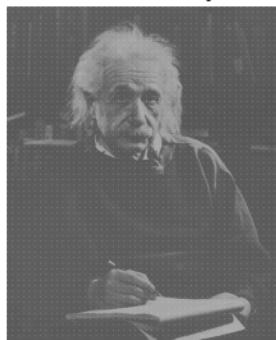
Film Grain Noisy



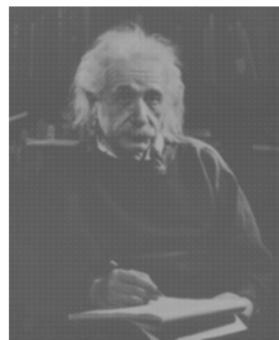
Film Grain Box Filtered



Periodic Noisy



Periodic Box Filtered



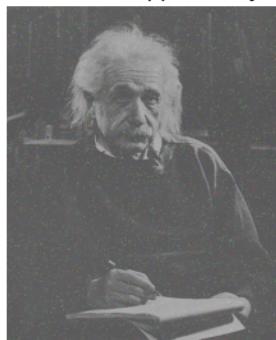
Speckle Noisy



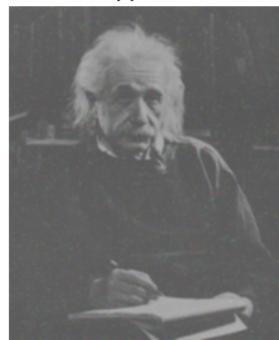
Speckle Box Filtered



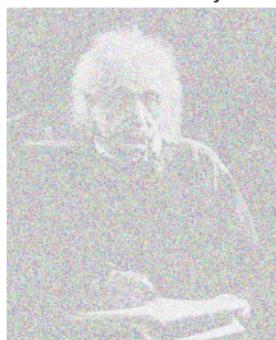
Salt-and-Pepper Noisy



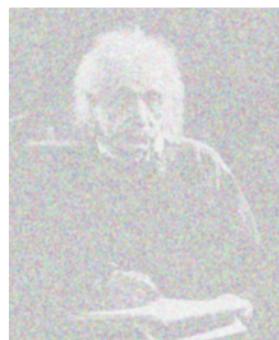
Salt-and-Pepper Box Filtered



Gaussian Noisy



Gaussian Box Filtered



1.2.2 Gaussian Filter

```
[7]: def apply_gaussian_filter(image):
    return cv2.GaussianBlur(image, (11, 11), sigmaX=1.5)

plt.figure(figsize=(8, 15))

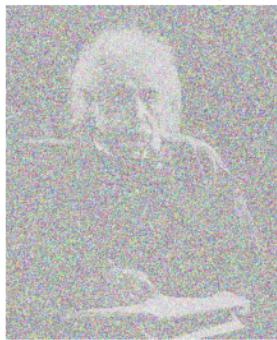
for i, (noise_type, noisy_image) in enumerate(noisy_images.items()):
    plt.subplot(len(noisy_images), 2, 2*i + 1)
    plt.imshow(cv2.cvtColor(noisy_image, cv2.COLOR_BGR2RGB))
    plt.title(noise_type + ' Noisy', fontsize=12)
    plt.axis('off')

    gaussian_filtered = apply_gaussian_filter(noisy_image)

    plt.subplot(len(noisy_images), 2, 2*i + 2)
    plt.imshow(cv2.cvtColor(gaussian_filtered, cv2.COLOR_BGR2RGB))
    plt.title(noise_type + ' Gaussian Filtered', fontsize=12)
    plt.axis('off')

plt.tight_layout()
plt.savefig('gaussian_filter_ppt.png', bbox_inches='tight')
plt.show()
```

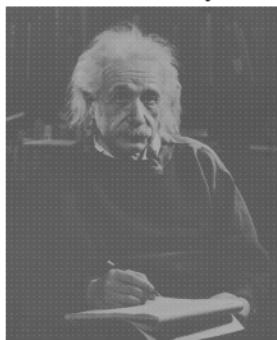
Film Grain Noisy



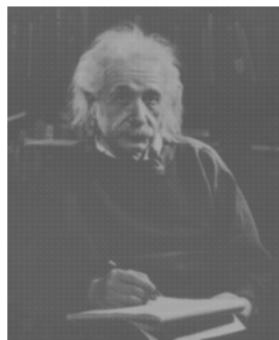
Film Grain Gaussian Filtered



Periodic Noisy



Periodic Gaussian Filtered



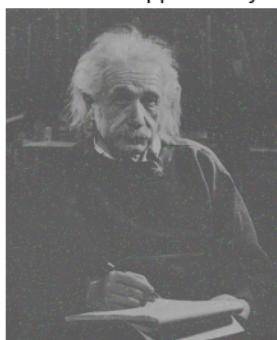
Speckle Noisy



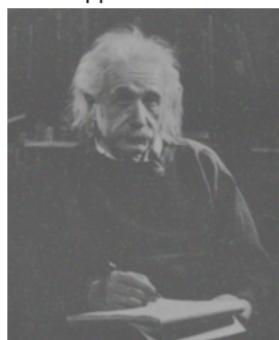
Speckle Gaussian Filtered



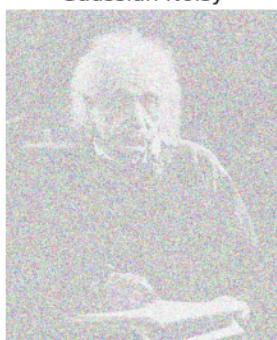
Salt-and-Pepper Noisy



Salt-and-Pepper Gaussian Filtered



Gaussian Noisy



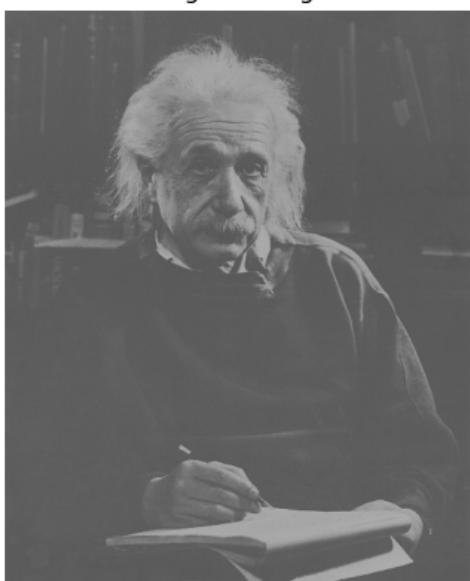
Gaussian Gaussian Filtered



1.2.3 Sobel

```
[8]: if original_image is None:  
    print("Error: Unable to load the image.")  
else:  
    sobel_x = cv2.Sobel(original_image, cv2.CV_64F, 1, 0, ksize=3)  
    sobel_y = cv2.Sobel(original_image, cv2.CV_64F, 0, 1, ksize=3)  
  
    sobel_mag = np.sqrt(sobel_x**2 + sobel_y**2)  
  
    threshold = 38  
    sobel_edges = np.where(sobel_mag > threshold, 255, 0).astype(np.uint8)  
  
    plt.figure(figsize=(10, 5))  
    plt.subplot(1, 2, 1)  
    plt.imshow(original_image, cmap='gray')  
    plt.title('Original Image')  
    plt.axis('off')  
  
    plt.subplot(1, 2, 2)  
    plt.imshow(sobel_edges, cmap='gray')  
    plt.title('Sobel Edges')  
    plt.axis('off')  
  
    plt.savefig('sobel_filter_ppt.png', bbox_inches='tight')  
    plt.show()
```

Original Image



Sobel Edges



```
[9]: def detect_edges(image):
    sobel_x = cv2.Sobel(image, cv2.CV_64F, 1, 0, ksize=3)
    sobel_y = cv2.Sobel(image, cv2.CV_64F, 0, 1, ksize=3)
    sobel_mag = np.sqrt(sobel_x**2 + sobel_y**2)
    threshold = 70
    sobel_edges = np.where(sobel_mag > threshold, 255, 0).astype(np.uint8)
    return sobel_edges

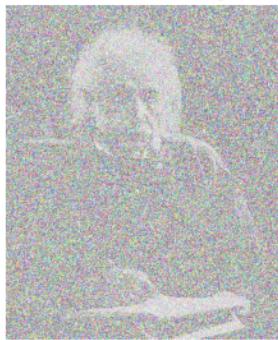
plt.figure(figsize=(8, 15))

for i, (noise_type, noisy_image) in enumerate(noisy_images.items()):
    plt.subplot(len(noisy_images), 2, 2*i + 1)
    plt.imshow(cv2.cvtColor(noisy_image, cv2.COLOR_BGR2RGB))
    plt.title(noise_type + ' Noisy', fontsize=12)
    plt.axis('off')

    plt.subplot(len(noisy_images), 2, 2*i + 2)
    edges = detect_edges(cv2.cvtColor(noisy_image, cv2.COLOR_BGR2GRAY))
    plt.imshow(edges, cmap='gray')
    plt.title(noise_type + ' Edges', fontsize=12)
    plt.axis('off')

plt.tight_layout()
plt.savefig('sobel_filter2_ppt.png', bbox_inches='tight')
plt.show()
```

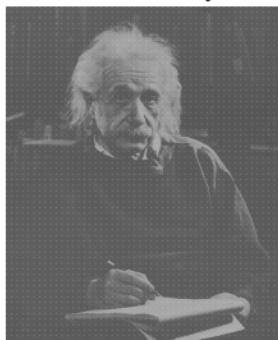
Film Grain Noisy



Film Grain Edges



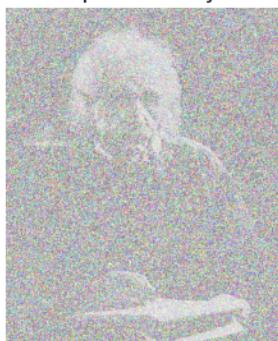
Periodic Noisy



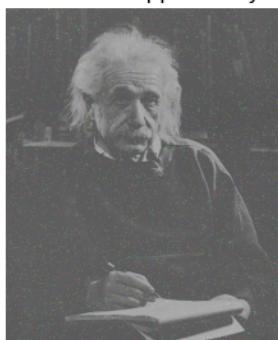
Periodic Edges



Speckle Noisy



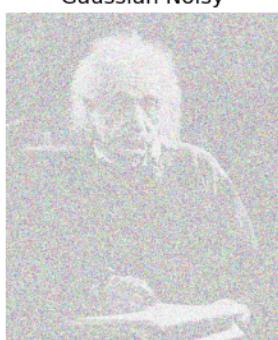
Salt-and-Pepper Noisy



Salt-and-Pepper Edges



Gaussian Noisy



1.3 Non-linear Filtering

1.3.1 Median Filter

```
[10]: def apply_median_filter(image):
    return cv2.medianBlur(image, 5)

plt.figure(figsize=(8, 15))

for i, (noise_type, noisy_image) in enumerate(noisy_images.items()):
    plt.subplot(len(noisy_images), 2, 2*i + 1)
    plt.imshow(cv2.cvtColor(noisy_image, cv2.COLOR_BGR2RGB))
    plt.title(noise_type + ' Noisy', fontsize=12)
    plt.axis('off')

    noisy_image_gray = cv2.cvtColor(noisy_image, cv2.COLOR_BGR2GRAY)
    median_filtered = apply_median_filter(noisy_image_gray)

    plt.subplot(len(noisy_images), 2, 2*i + 2)
    plt.imshow(median_filtered, cmap='gray')
    plt.title(noise_type + ' Median Filtered', fontsize=12)
    plt.axis('off')

plt.tight_layout()
plt.savefig('median_filter_ppt.png', bbox_inches='tight')
plt.show()
```

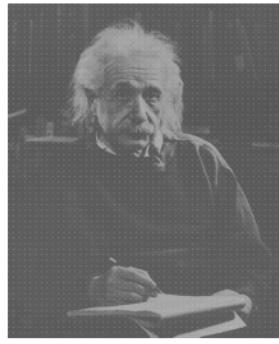
Film Grain Noisy



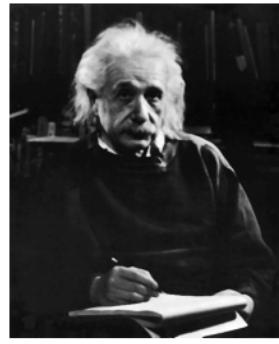
Film Grain Median Filtered



Periodic Noisy



Periodic Median Filtered



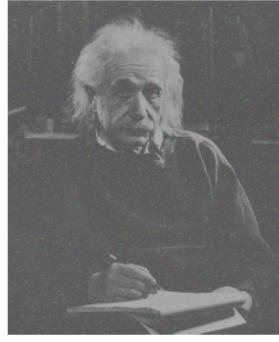
Speckle Noisy



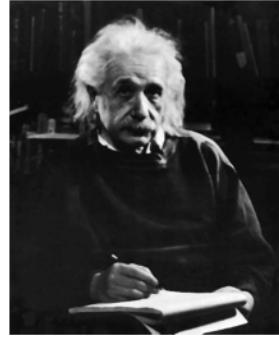
Speckle Median Filtered



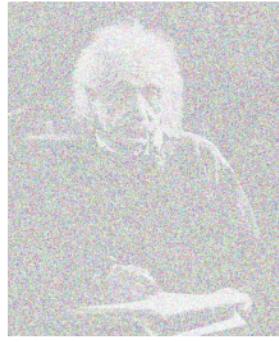
Salt-and-Pepper Noisy



Salt-and-Pepper Median Filtered



Gaussian Noisy



Gaussian Median Filtered



1.3.2 Rank Filter

```
[11]: def apply_rank_filter(image):
    kernel_size = 3
    rank = 4

    padded_image = cv2.copyMakeBorder(image, 1, 1, 1, 1, cv2.BORDER_REPLICATE)
    filtered_image = np.zeros_like(image)

    for y in range(image.shape[0]):
        for x in range(image.shape[1]):
            neighborhood = padded_image[y:y+kernel_size, x:x+kernel_size]
            sorted_neighborhood = np.sort(neighborhood.flatten())
            filtered_image[y, x] = sorted_neighborhood[rank]

    return filtered_image

num_noises = len(noisy_images)
num_images = num_noises

plt.figure(figsize=(8, 15))

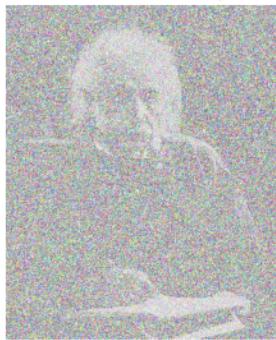
for i, (noise_type, noisy_image) in enumerate(noisy_images.items()):
    plt.subplot(len(noisy_images), 2, 2*i + 1)
    plt.imshow(cv2.cvtColor(noisy_image, cv2.COLOR_BGR2RGB))
    plt.title(noise_type + ' Noisy', fontsize=12)
    plt.axis('off')

    noisy_image_gray = cv2.cvtColor(noisy_image, cv2.COLOR_BGR2GRAY)
    rank_filtered = apply_rank_filter(noisy_image_gray)

    plt.subplot(len(noisy_images), 2, 2*i + 2)
    plt.imshow(rank_filtered, cmap='gray')
    plt.title(noise_type + ' Rank Filtered', fontsize=12)
    plt.axis('off')

plt.tight_layout()
plt.savefig('rank_filter_ppt.png', bbox_inches='tight')
plt.show()
```

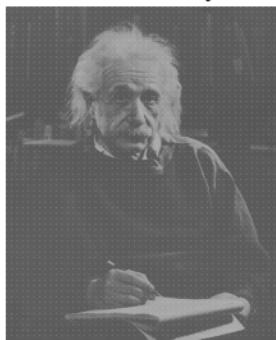
Film Grain Noisy



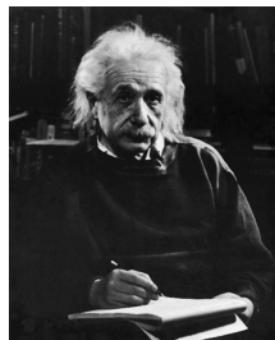
Film Grain Rank Filtered



Periodic Noisy



Periodic Rank Filtered



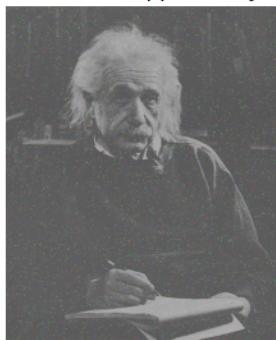
Speckle Noisy



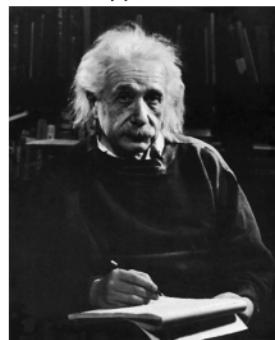
Speckle Rank Filtered



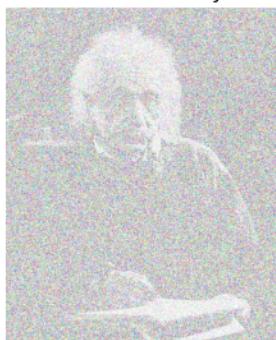
Salt-and-Pepper Noisy



Salt-and-Pepper Rank Filtered



Gaussian Noisy



Gaussian Rank Filtered



1.3.3 Adaptive Filter

```
[12]: def apply_adaptive_filter(image):
    gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    return cv2.adaptiveThreshold(gray_image, 255, cv2.
        ↪ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY, 11, 2)

num_noises = len(noisy_images)
num_images = num_noises + 1

plt.figure(figsize=(8, 15))

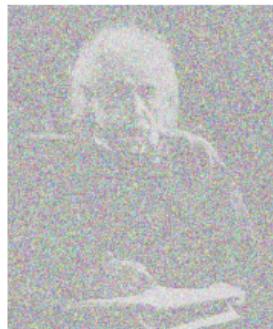
for i, (noise_type, noisy_image) in enumerate(noisy_images.items(), start=1):
    plt.subplot(num_images, 2, i * 2 - 1)
    plt.imshow(cv2.cvtColor(noisy_image, cv2.COLOR_BGR2RGB))
    plt.title(noise_type + ' Noisy')
    plt.axis('off')

    adaptive_filtered = apply_adaptive_filter(noisy_image)

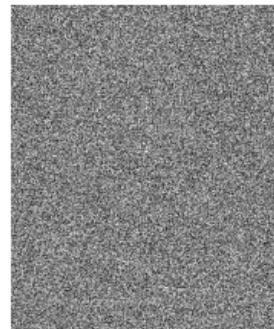
    plt.subplot(num_images, 2, i * 2)
    plt.imshow(adaptive_filtered, cmap='gray')
    plt.title(noise_type + ' Adaptive Filtered')
    plt.axis('off')

plt.tight_layout()
plt.savefig('adaptive_filter_ppt.png', bbox_inches='tight')
plt.show()
```

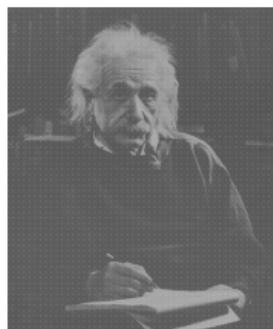
Film Grain Noisy



Film Grain Adaptive Filtered



Periodic Noisy



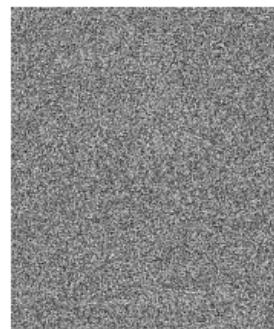
Periodic Adaptive Filtered



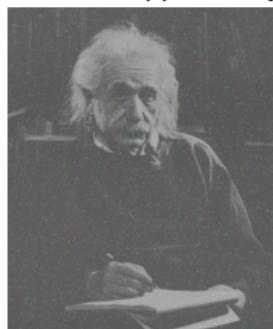
Speckle Noisy



Speckle Adaptive Filtered



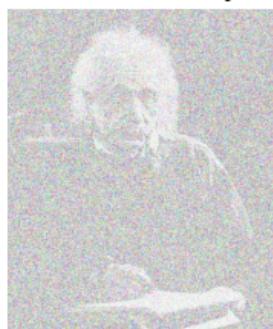
Salt-and-Pepper Noisy



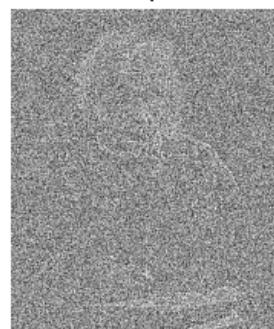
Salt-and-Pepper Adaptive Filtered



Gaussian Noisy



Gaussian Adaptive Filtered



1.3.4 Bilateral Filter

```
[13]: def apply_bilateral_filter(image):
    return cv2.bilateralFilter(image, 9, 75, 75)

num_noises = len(noisy_images)
num_images = num_noises + 1

plt.figure(figsize=(8, 15))

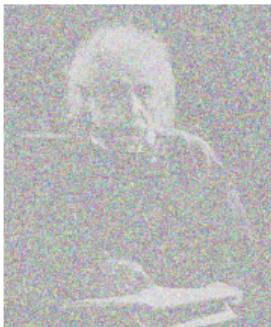
for i, (noise_type, noisy_image) in enumerate(noisy_images.items(), start=1):
    plt.subplot(num_images, 2, i * 2 + 1)
    plt.imshow(cv2.cvtColor(noisy_image, cv2.COLOR_BGR2RGB))
    plt.title(noise_type + ' Noisy')
    plt.axis('off')

    bilateral_filtered = apply_bilateral_filter(noisy_image)

    plt.subplot(num_images, 2, i * 2 + 2)
    plt.imshow(cv2.cvtColor(bilateral_filtered, cv2.COLOR_BGR2RGB))
    plt.title(noise_type + ' Bilateral Filtered')
    plt.axis('off')

plt.tight_layout()
plt.savefig('bilateral_filter_ppt.png', bbox_inches='tight')
plt.show()
```

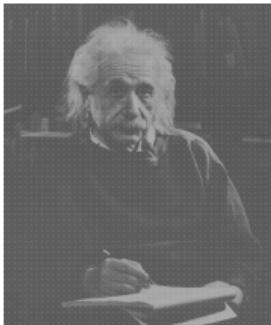
Film Grain Noisy



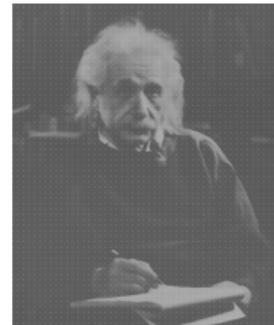
Film Grain Bilateral Filtered



Periodic Noisy



Periodic Bilateral Filtered



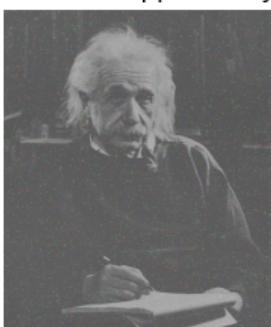
Speckle Noisy



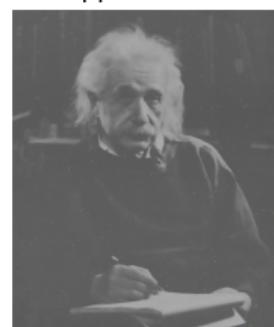
Speckle Bilateral Filtered



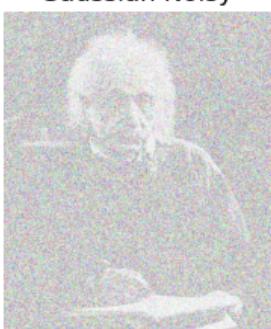
Salt-and-Pepper Noisy



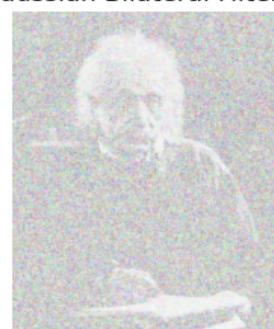
Salt-and-Pepper Bilateral Filtered



Gaussian Noisy



Gaussian Bilateral Filtered



2 Image Filtering on colour image

```
[14]: image_path = "I:/My Drive/Image_filtering/images/images/animated1.jpg"
# image_path = "/Users/user/Downloads/images/animated1.jpg"
bgr_image = cv2.imread(image_path)
original_image = cv2.cvtColor(bgr_image, cv2.COLOR_BGR2RGB)
```

```
[15]: plt.imshow(original_image)
plt.axis('off')
plt.show()
```



```
[16]: if original_image is None:
    print("Error: Unable to load the image.")
else:
    def generate_film_grain(image):
        film_grain_noise = np.random.normal(loc=1, scale=1, size=image.shape).
        ↪astype(np.uint8)
        return cv2.add(image, film_grain_noise)

    def generate_periodic(image):
        periodic_noise = np.zeros(image.shape, dtype=np.uint8)
        periodic_noise[::5, ::5, :] = 255
```

```

    return cv2.add(image, periodic_noise)

def generate_speckle(image):
    speckle_noise = np.random.normal(loc=1, scale=0.5, size=image.shape) * 255
    return cv2.add(image, speckle_noise.astype(np.uint8))

def generate_salt_and_pepper(image):
    salt_pepper_noise = np.random.choice([0, 255], size=image.shape[:2] + (image.shape[2],), p=[0.89, 0.11]).astype(np.uint8)
    return cv2.add(image, salt_pepper_noise)

def generate_gaussian(image):
    gaussian_noise = np.random.normal(loc=1, scale=2, size=image.shape).astype(np.uint8)
    return cv2.add(image, gaussian_noise)

noise_generators = {
    'Film Grain': generate_film_grain,
    'Periodic': generate_periodic,
    'Speckle': generate_speckle,
    'Salt-and-Pepper': generate_salt_and_pepper,
    'Gaussian': generate_gaussian
}

noisy_images = {noise_type: noise_generator(original_image) for noise_type, noise_generator in noise_generators.items()}

```

```
[17]: plt.figure(figsize=(15, 15))
for i, (noise_type, noisy_image) in enumerate(noisy_images.items(), start=1):
    plt.subplot(3, 2, i)
    plt.imshow(noisy_image)
    plt.title(noise_type)
    plt.axis('off')
plt.show()
```



2.1 Linear Filtering

2.1.1 BOX Filter

```
[18]: def apply_box_filter(image):
        return cv2.boxFilter(image, -1, (5, 5))

plt.figure(figsize=(8, 15))

for noise_type, noisy_image in noisy_images.items():
    plt.subplot(len(noisy_images), 2, 2*(list(noisy_images.keys())->
        index(noise_type)) + 1)
```

```
plt.imshow(noisy_image)
plt.title(noise_type + ' Noisy', fontsize=12)
plt.axis('off')

box_filtered = apply_box_filter(noisy_image)

plt.subplot(len(noisy_images), 2, 2*(list(noisy_images.keys()).
    index(noise_type)) + 2)
plt.imshow(box_filtered)
plt.title(noise_type + ' Box Filtered', fontsize=12)
plt.axis('off')

plt.tight_layout()
plt.savefig('box_filter_output.png', bbox_inches='tight')
plt.show()
```

Film Grain Noisy



Film Grain Box Filtered



Periodic Noisy



Periodic Box Filtered



Speckle Noisy



Speckle Box Filtered



Salt-and-Pepper Noisy



Salt-and-Pepper Box Filtered



Gaussian Noisy



Gaussian Box Filtered



2.1.2 Gaussian Filter

```
[19]: def apply_gaussian_filter(image):
    return cv2.GaussianBlur(image, (11, 11), sigmaX=1.5)

plt.figure(figsize=(8, 15))

for i, (noise_type, noisy_image) in enumerate(noisy_images.items()):
    plt.subplot(len(noisy_images), 2, 2*i + 1)
    plt.imshow(noisy_image)
    plt.title(noise_type + ' Noisy', fontsize=12)
    plt.axis('off')

    gaussian_filtered = apply_gaussian_filter(noisy_image)

    plt.subplot(len(noisy_images), 2, 2*i + 2)
    plt.imshow(gaussian_filtered)
    plt.title(noise_type + ' Gaussian Filtered', fontsize=12)
    plt.axis('off')

plt.tight_layout()
plt.savefig('gaussian_filter_output.png', bbox_inches='tight')
plt.show()
```

Film Grain Noisy



Film Grain Gaussian Filtered



Periodic Noisy



Periodic Gaussian Filtered



Speckle Noisy



Speckle Gaussian Filtered



Salt-and-Pepper Noisy



Salt-and-Pepper Gaussian Filtered



Gaussian Noisy



Gaussian Gaussian Filtered



2.1.3 Sobel

```
[20]: if original_image is None:  
    print("Error: Unable to load the image.")  
else:  
    sobel_x = cv2.Sobel(original_image, cv2.CV_64F, 1, 0, ksize=3)  
    sobel_y = cv2.Sobel(original_image, cv2.CV_64F, 0, 1, ksize=3)  
  
    sobel_mag = np.sqrt(sobel_x**2 + sobel_y**2)  
  
    threshold = 130  
    sobel_edges = np.where(sobel_mag > threshold, 255, 0).astype(np.uint8)  
  
    plt.figure(figsize=(10, 5))  
    plt.subplot(1, 2, 1)  
    plt.imshow(original_image, cmap='gray')  
    plt.title('Original Image')  
    plt.axis('off')  
  
    plt.subplot(1, 2, 2)  
    plt.imshow(sobel_edges, cmap='gray')  
    plt.title('Sobel Edges')  
    plt.axis('off')  
    plt.savefig('sobel_filter_output.png', bbox_inches='tight')  
    plt.show()
```



```
[21]: def detect_edges(image):  
    sobel_x = cv2.Sobel(image, cv2.CV_64F, 1, 0, ksize=3)  
    sobel_y = cv2.Sobel(image, cv2.CV_64F, 0, 1, ksize=3)  
    sobel_mag = np.sqrt(sobel_x**2 + sobel_y**2)
```

```

threshold = 255
sobel_edges = np.where(sobel_mag > threshold, 255, 0).astype(np.uint8)
return sobel_edges

plt.figure(figsize=(8, 15))

for i, (noise_type, noisy_image) in enumerate(noisy_images.items()):
    plt.subplot(len(noisy_images), 2, 2*i + 1)
    plt.imshow(noisy_image)
    plt.title(noise_type + ' Noisy', fontsize=12)
    plt.axis('off')

    plt.subplot(len(noisy_images), 2, 2*i + 2)
    edges = detect_edges(noisy_image)
    plt.imshow(edges, cmap='gray')
    plt.title(noise_type + ' Edges', fontsize=12)
    plt.axis('off')

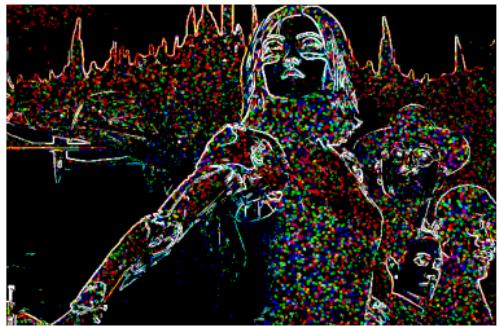
plt.tight_layout()
plt.savefig('sobel_filter2_output.png', bbox_inches='tight')
plt.show()

```

Film Grain Noisy



Film Grain Edges



Periodic Noisy



Periodic Edges



Speckle Noisy



Salt-and-Pepper Noisy



Salt-and-Pepper Edges



Gaussian Noisy



Gaussian Edges



2.2 Non-linear Filtering

2.2.1 Median Filter

```
[22]: def apply_median_filter(image):
    return cv2.medianBlur(image, 5)

plt.figure(figsize=(8, 15))

for i, (noise_type, noisy_image) in enumerate(noisy_images.items()):
    plt.subplot(len(noisy_images), 2, 2*i + 1)
    plt.imshow(noisy_image)
    plt.title(noise_type + ' Noisy', fontsize=12)
    plt.axis('off')

# noisy_image_gray = cv2.cvtColor(noisy_image, cv2.COLOR_BGR2GRAY)
median_filtered = apply_median_filter(noisy_image)

plt.subplot(len(noisy_images), 2, 2*i + 2)
plt.imshow(median_filtered, cmap='gray')
plt.title(noise_type + ' Median Filtered', fontsize=12)
plt.axis('off')

plt.tight_layout()
plt.savefig('median_filter_output.png', bbox_inches='tight')
plt.show()
```

Film Grain Noisy



Film Grain Median Filtered



Periodic Noisy



Periodic Median Filtered



Speckle Noisy



Speckle Median Filtered



Salt-and-Pepper Noisy



Salt-and-Pepper Median Filtered



Gaussian Noisy



Gaussian Median Filtered



2.2.2 Rank Filter

```
[23]: def apply_rank_filter(image):
    kernel_size = 3
    rank = 4

    padded_image = cv2.copyMakeBorder(image, 1, 1, 1, 1, cv2.BORDER_REPLICATE)
    filtered_image = np.zeros_like(image)

    for y in range(image.shape[0]):
        for x in range(image.shape[1]):
            neighborhood = padded_image[y:y+kernel_size, x:x+kernel_size]
            sorted_neighborhood = np.sort(neighborhood.flatten())
            filtered_image[y, x] = sorted_neighborhood[rank]

    return filtered_image

num_noises = len(noisy_images)
num_images = num_noises

plt.figure(figsize=(8, 15))

for i, (noise_type, noisy_image) in enumerate(noisy_images.items()):
    plt.subplot(len(noisy_images), 2, 2*i + 1)
    plt.imshow(noisy_image)
    plt.title(noise_type + ' Noisy', fontsize=12)
    plt.axis('off')

    # noisy_image_gray = cv2.cvtColor(noisy_image, cv2.COLOR_BGR2GRAY)
    rank_filtered = apply_rank_filter(noisy_image)

    plt.subplot(len(noisy_images), 2, 2*i + 2)
    plt.imshow(rank_filtered)
    plt.title(noise_type + ' Rank Filtered', fontsize=12)
    plt.axis('off')

plt.tight_layout()
plt.savefig('rank_filter_output.png', bbox_inches='tight')
plt.show()
```

Film Grain Noisy



Film Grain Rank Filtered



Periodic Noisy



Periodic Rank Filtered



Speckle Noisy



Speckle Rank Filtered



Salt-and-Pepper Noisy



Salt-and-Pepper Rank Filtered



Gaussian Noisy



Gaussian Rank Filtered



2.2.3 Adaptive Filter

```
[24]: def apply_adaptive_filter(image):
    gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    return cv2.adaptiveThreshold(gray_image, 255, cv2.
        ↪ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY, 11, 2)

num_noises = len(noisy_images)
num_images = num_noises + 1

plt.figure(figsize=(8, 15))

for i, (noise_type, noisy_image) in enumerate(noisy_images.items(), start=1):
    plt.subplot(num_images, 2, i * 2 - 1)
    plt.imshow(noisy_image)
    plt.title(noise_type + ' Noisy')
    plt.axis('off')

    adaptive_filtered = apply_adaptive_filter(noisy_image)

    plt.subplot(num_images, 2, i * 2)
    plt.imshow(adaptive_filtered, cmap='gray')
    plt.title(noise_type + ' Adaptive Filtered')
    plt.axis('off')

plt.tight_layout()
plt.savefig('adaptive_filter_output.png', bbox_inches='tight')
plt.show()
```

Film Grain Noisy



Film Grain Adaptive Filtered



Periodic Noisy



Periodic Adaptive Filtered



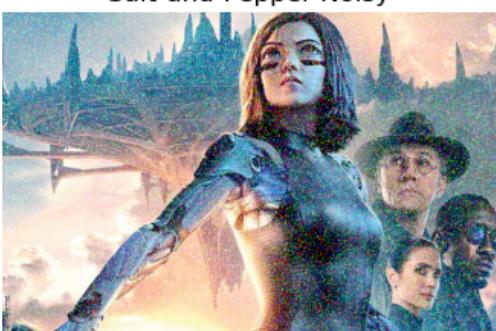
Speckle Noisy



Speckle Adaptive Filtered



Salt-and-Pepper Noisy



Salt-and-Pepper Adaptive Filtered



Gaussian Noisy



Gaussian Adaptive Filtered



2.2.4 Bilateral Filter

```
[25]: def apply_bilateral_filter(image):
    return cv2.bilateralFilter(image, 9, 75, 75)

num_noises = len(noisy_images)
num_images = num_noises + 1

plt.figure(figsize=(8, 15))

for i, (noise_type, noisy_image) in enumerate(noisy_images.items(), start=1):
    plt.subplot(num_images, 2, i * 2 + 1)
    plt.imshow(noisy_image)
    plt.title(noise_type + ' Noisy')
    plt.axis('off')

    bilateral_filtered = apply_bilateral_filter(noisy_image)

    plt.subplot(num_images, 2, i * 2 + 2)
    plt.imshow(bilateral_filtered)
    plt.title(noise_type + ' Bilateral Filtered')
    plt.axis('off')

plt.tight_layout()
# Save the generated output
plt.savefig('output.png', bbox_inches='tight')
plt.savefig('bilateral_filter_output.png', bbox_inches='tight')
plt.show()
```

Film Grain Noisy



Film Grain Bilateral Filtered



Periodic Noisy



Periodic Bilateral Filtered



Speckle Noisy



Speckle Bilateral Filtered



Salt-and-Pepper Noisy



Salt-and-Pepper Bilateral Filtered



Gaussian Noisy



Gaussian Bilateral Filtered



3 Gray Scale Imaging

3.1 Average Method

```
[26]: def average_method(img):
    grayscale_img = np.mean(img, axis=2)

    grayscale_img = np.uint8(grayscale_img)

    return grayscale_img

color_img = cv2.imread("I:/My Drive/Image_filtering/images/images/flower1.jpg")

grayscale_img_avg = average_method(color_img)

org = cv2.cvtColor(color_img, cv2.COLOR_BGR2RGB)

plt.figure(figsize=(15, 15))

plt.subplot(1,2,1)
plt.imshow(org)
plt.title('Color Image')
plt.axis('off') # Hide axis

plt.subplot(1,2,2)
plt.imshow(grayscale_img_avg, cmap='gray') # Specify grayscale colormap
plt.title('Grayscale Image (Average Method)')
plt.axis('off') # Hide axis

plt.show()
```



3.2 Luminosity Method

```
[27]: def luminosity_method(img):
    R = img[:, :, 0]
    G = img[:, :, 1]
    B = img[:, :, 2]

    grayscale_img = 0.21 * R + 0.72 * G + 0.07 * B
    grayscale_img = np.uint8(grayscale_img)

    return grayscale_img

color_img = cv2.imread("I:/My Drive/Image_filtering/images/images/flower.jpg")

# Convert the color image to grayscale using the luminosity method
grayscale_img_lum = luminosity_method(color_img)

org = cv2.cvtColor(color_img, cv2.COLOR_BGR2RGB)

plt.figure(figsize=(15, 15))

plt.subplot(1, 2, 1)
plt.imshow(org)
plt.title('Color Image')
plt.axis('off') # Hide axis

plt.subplot(1, 2, 2)
plt.imshow(grayscale_img_lum, cmap='gray') # Specify grayscale colormap
plt.title('Grayscale Image (Luminosity Method)')
plt.axis('off') # Hide axis

plt.show()
```



3.3 Single Channel Method

```
[28]: color_img = cv2.imread("I:/My Drive/Image_filtering/images/images/animated2.  
↪jpg")  
  
green_channel = color_img[:, :, 1]  
  
# Extract the red channel  
red_channel = color_img[:, :, 2]  
  
# Extract the blue channel  
blue_channel = color_img[:, :, 0]  
  
org = cv2.cvtColor(color_img, cv2.COLOR_BGR2RGB)  
  
plt.figure(figsize=(15, 15))  
  
plt.subplot(2,2,1)  
plt.imshow(org)  
plt.title('Color Image')  
plt.axis('off') # Hide axis  
  
plt.subplot(2,2,2)  
plt.imshow(green_channel, cmap='gray') # Specify grayscale colormap  
plt.title('Grayscale Image (Single Channel - Green)')  
plt.axis('off') # Hide axis  
  
plt.subplot(2,2,3)  
plt.imshow(red_channel, cmap='gray') # Specify grayscale colormap  
plt.title('Grayscale Image (Single Channel - Red)')  
plt.axis('off') # Hide axis  
  
plt.subplot(2,2,4)  
plt.imshow(blue_channel, cmap='gray') # Specify grayscale colormap  
plt.title('Grayscale Image (Single Channel - Blue)')  
plt.axis('off') # Hide axis  
  
plt.show()
```

Color Image



Grayscale Image (Single Channel - Green)



Grayscale Image (Single Channel - Red)



Grayscale Image (Single Channel - Blue)



4 Image Blurring

```
[29]: image_path = "I:/My Drive/Image_filtering/images/images/animated1.jpg"
# image_path = "/Users/user/Downloads/images/animated1.jpg"
bgr_image = cv2.imread(image_path)
img = cv2.cvtColor(bgr_image, cv2.COLOR_BGR2RGB)
```

```
plt.imshow(img)
plt.axis('off') # Hide axis
plt.show()
```



```
[30]: averaging_blur = cv2.blur(img, (11, 11))
gaussian_blur = cv2.GaussianBlur(img, (11, 11), 0)
bilateral_blur = cv2.bilateralFilter(img, 9, 75, 75)
median_blur = cv2.medianBlur(img, 11)
```

```
[31]: display_size = (200, 200)
original_display = cv2.resize(img, display_size)
```

```
[32]: plt.figure(figsize=(10, 10))

plt.subplot(2,2,1)
plt.imshow(averaging_blur)
plt.title('averaging blur')
plt.axis('off') # Hide axis

plt.subplot(2,2,2)
plt.imshow(gaussian_blur)
plt.title('gaussian blur')
plt.axis('off') # Hide axis

plt.subplot(2,2,3)
```

```

plt.imshow(bilateral_blur)
plt.title('bilateral blur')
plt.axis('off')

plt.subplot(2,2,4)
plt.imshow(median_blur)
plt.title('median blur')
plt.axis('off')

plt.tight_layout()
plt.show()

```



5 Histogram based image analysis

```

[33]: image_path = "I:/My Drive/Image_filtering/images/images/
        ↪Fig0310(b) (washed_out_pollen_image).tif"
image = Image.open(image_path).convert('L')

```

```
[34]: def histogram_equalization(image):
    # Calculate histogram
    hist, bins = np.histogram(image.flatten(), 256, [0,256])

    # Calculate cumulative distribution function
    cdf = hist.cumsum()
    cdf_normalized = cdf * hist.max() / cdf.max()

    # Perform histogram equalization
    cdf_m = np.ma.masked_equal(cdf, 0)
    cdf_m = (cdf_m - cdf_m.min())*255 / (cdf_m.max()-cdf_m.min())
    cdf = np.ma.filled(cdf_m, 0).astype('uint8')

    # Apply histogram equalization
    equalized_image = cdf[image]

    return equalized_image, hist
```

```
[35]: equalized_image, hist = histogram_equalization(np.array(image))
```

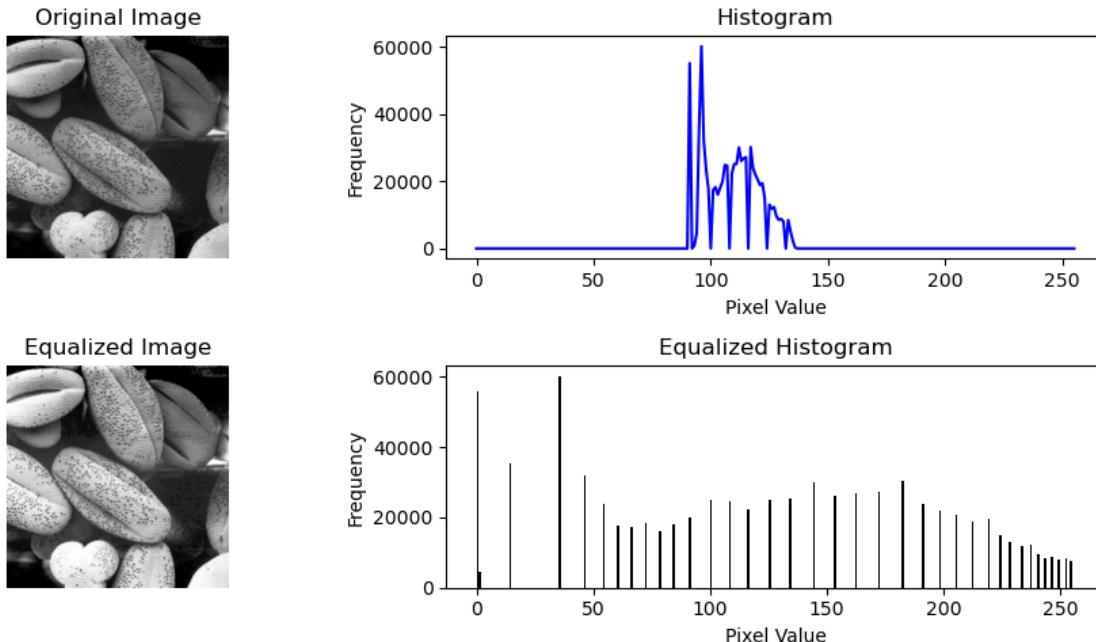
```
[36]: plt.figure(figsize=(10, 5))
plt.subplot(2, 2, 1)
plt.imshow(image, cmap='gray')
plt.title('Original Image')
plt.axis('off')

plt.subplot(2, 2, 3)
plt.imshow(equalized_image, cmap='gray')
plt.title('Equalized Image')
plt.axis('off')

# Display histogram
plt.subplot(2, 2, 2)
plt.plot(hist, color='blue')
plt.title('Histogram')
plt.xlabel('Pixel Value')
plt.ylabel('Frequency')

plt.subplot(2, 2, 4)
plt.hist(equalized_image.flatten(), 256, [0,256], color='black')
plt.title('Equalized Histogram')
plt.xlabel('Pixel Value')
plt.ylabel('Frequency')

plt.tight_layout()
plt.show()
```



```
[37]: image_path = "I:/My Drive/Image_filtering/images/images/  
    ↴Fig0354(a)(einstein_orig).tif"  
    image = Image.open(image_path).convert('L')
```

```
[38]: def histogram_equalization(image):
    # Calculate histogram
    hist, bins = np.histogram(image.flatten(), 256, [0,256])

    # Calculate cumulative distribution function
    cdf = hist.cumsum()
    cdf_normalized = cdf * hist.max() / cdf.max()

    # Perform histogram equalization
    cdf_m = np.ma.masked_equal(cdf, 0)
    cdf_m = (cdf_m - cdf_m.min())*255 / (cdf_m.max()-cdf_m.min())
    cdf = np.ma.filled(cdf_m, 0).astype('uint8')

    # Apply histogram equalization
    equalized_image = cdf[image]

    return equalized_image, hist, cdf
```

```
[39]: equalized_image, hist_original, cdf_equalized = histogram_equalization(np.array(image))
```

```
[40]: plt.figure(figsize=(12, 6))

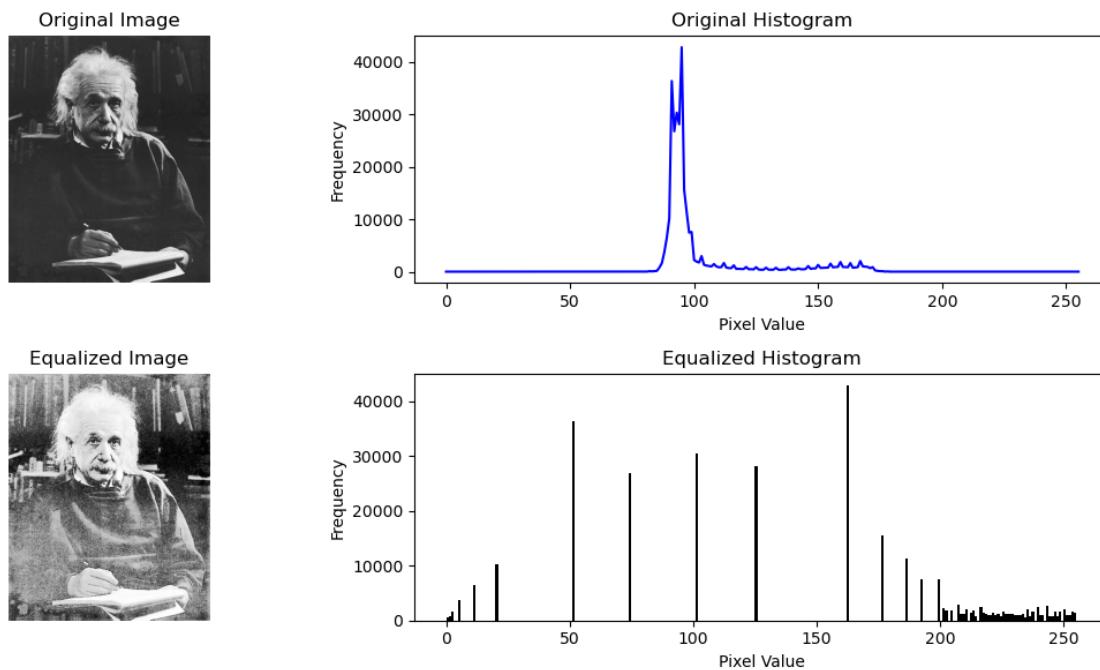
plt.subplot(2, 2, 1)
plt.imshow(image, cmap='gray')
plt.title('Original Image')
plt.axis('off')

plt.subplot(2, 2, 3)
plt.imshow(equalized_image, cmap='gray')
plt.title('Equalized Image')
plt.axis('off')

# Display histograms
plt.subplot(2, 2, 2)
plt.plot(hist_original, color='blue')
plt.title('Original Histogram')
plt.xlabel('Pixel Value')
plt.ylabel('Frequency')

plt.subplot(2, 2, 4)
plt.hist(equalized_image.flatten(), 256, [0,256], color='black')
plt.title('Equalized Histogram')
plt.xlabel('Pixel Value')
plt.ylabel('Frequency')

plt.tight_layout()
plt.show()
```



6 Image Scaling

6.1 Nearest neighbor

```
[41]: from PIL import Image, ImageDraw, ImageFont
from IPython.display import display

[42]: # original_img = Image.open("/Users/user/Downloads/animated2.jpg")
original_img = Image.open("I:/My Drive/Image_filtering/images/images/animated2.
↪jpg")

original_width, original_height = original_img.size

new_width = original_width // 2
new_height = original_height // 2

[43]: resized_img = original_img.resize((new_width, new_height), resample=Image.
↪NEAREST)

[44]: # plt.figure(figsize=(12, 6))

# plt.subplot(1, 2, 1)
# plt.imshow(original_img)
# plt.title('Original Image')

# plt.subplot(1, 2, 2)
# plt.imshow(resized_img)
# plt.title('Equalized Image')

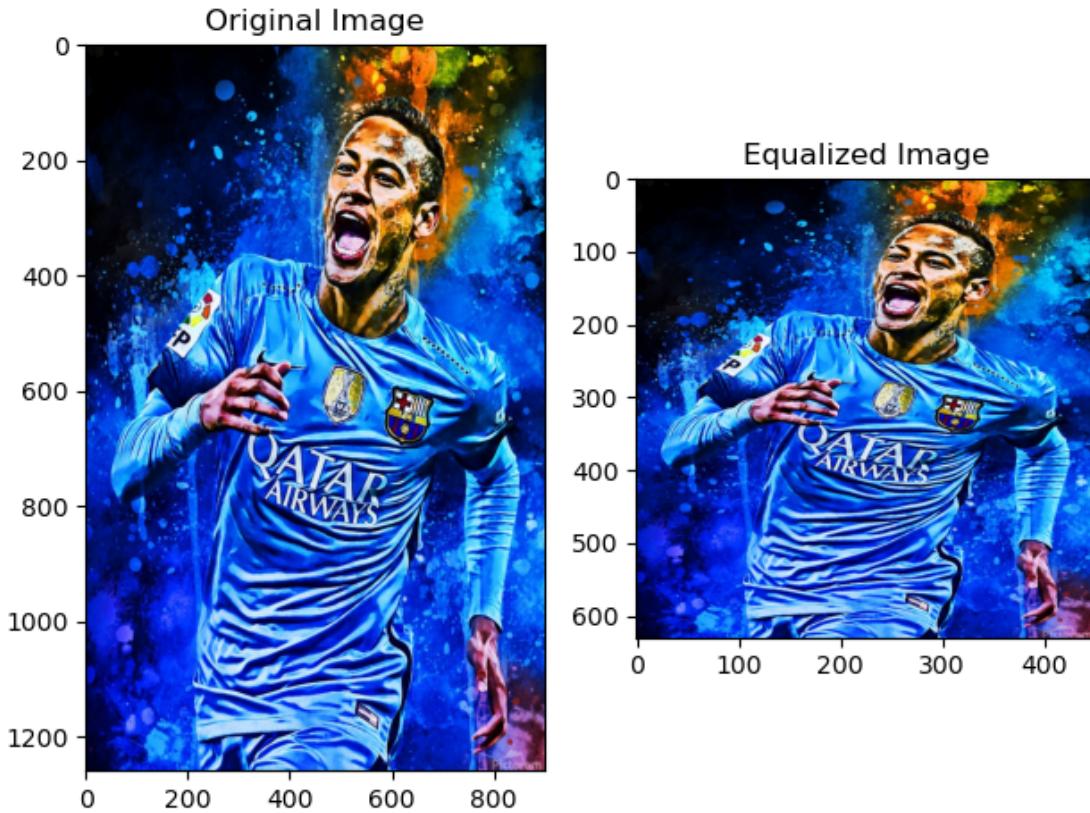
# plt.tight_layout()
# plt.show()

[45]: original_width, original_height = original_img.size

plt.subplot(1, 2, 1)
plt.imshow(original_img)
plt.title('Original Image')
plt.gca().set_aspect('auto') # Ensure the aspect ratio is automatic

plt.subplot(1, 2, 2)
plt.imshow(resized_img)
plt.title('Equalized Image')
plt.gca().set_aspect(original_width/original_height) # Set aspect ratio based
↪on original image dimensions
```

```
plt.tight_layout()  
plt.show()
```



6.2 Bilinar Interpolation

```
[46]: resized_img = original_img.resize((new_width, new_height), resample=Image.  
    ↪BILINEAR)
```

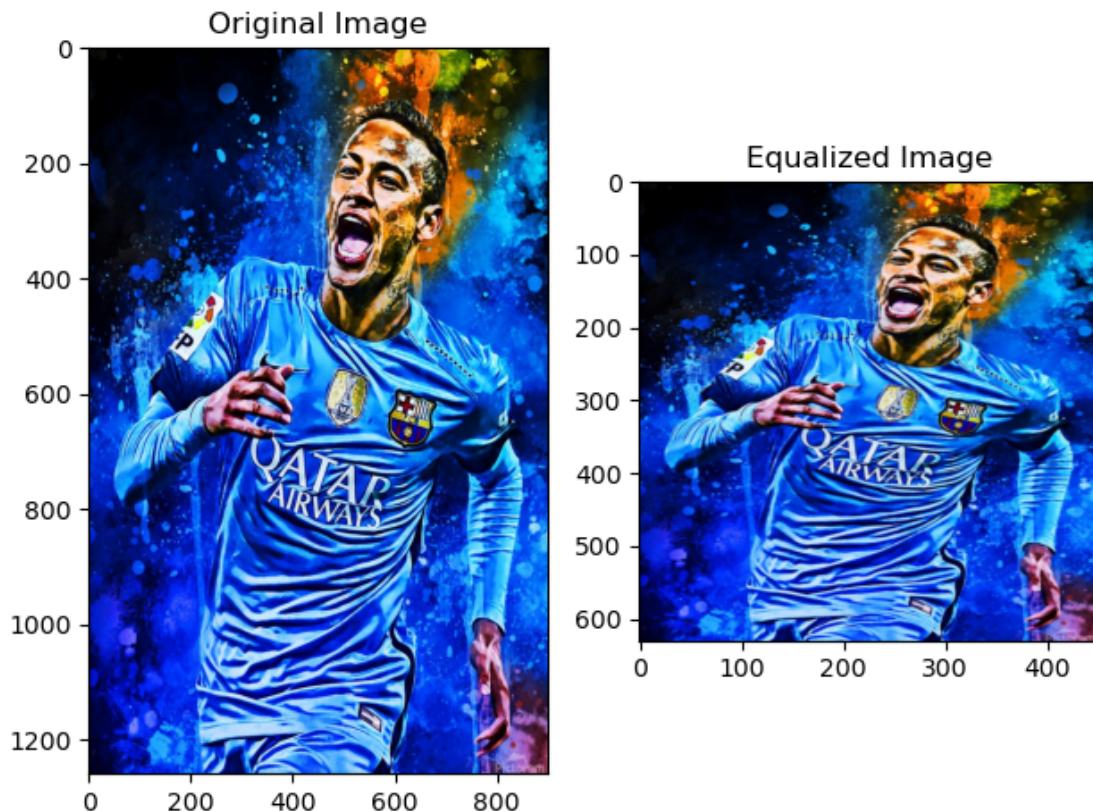
```
[47]: original_width, original_height = original_img.size  
  
plt.subplot(1, 2, 1)  
plt.imshow(original_img)  
plt.title('Original Image')  
plt.gca().set_aspect('auto') # Ensure the aspect ratio is automatic  
  
plt.subplot(1, 2, 2)  
plt.imshow(resized_img)  
plt.title('Equalized Image')
```

```

plt.gca().set_aspect(original_width/original_height) # Set aspect ratio based
# on original image dimensions

plt.tight_layout()
plt.show()

```



6.3 Bicubic Interpolation

[48]: `resized_img = original_img.resize((new_width, new_height), resample=Image.BICUBIC)`

[49]: `original_width, original_height = original_img.size`

```

plt.subplot(1, 2, 1)
plt.imshow(original_img)
plt.title('Original Image')
plt.gca().set_aspect('auto') # Ensure the aspect ratio is automatic

plt.subplot(1, 2, 2)
plt.imshow(resized_img)

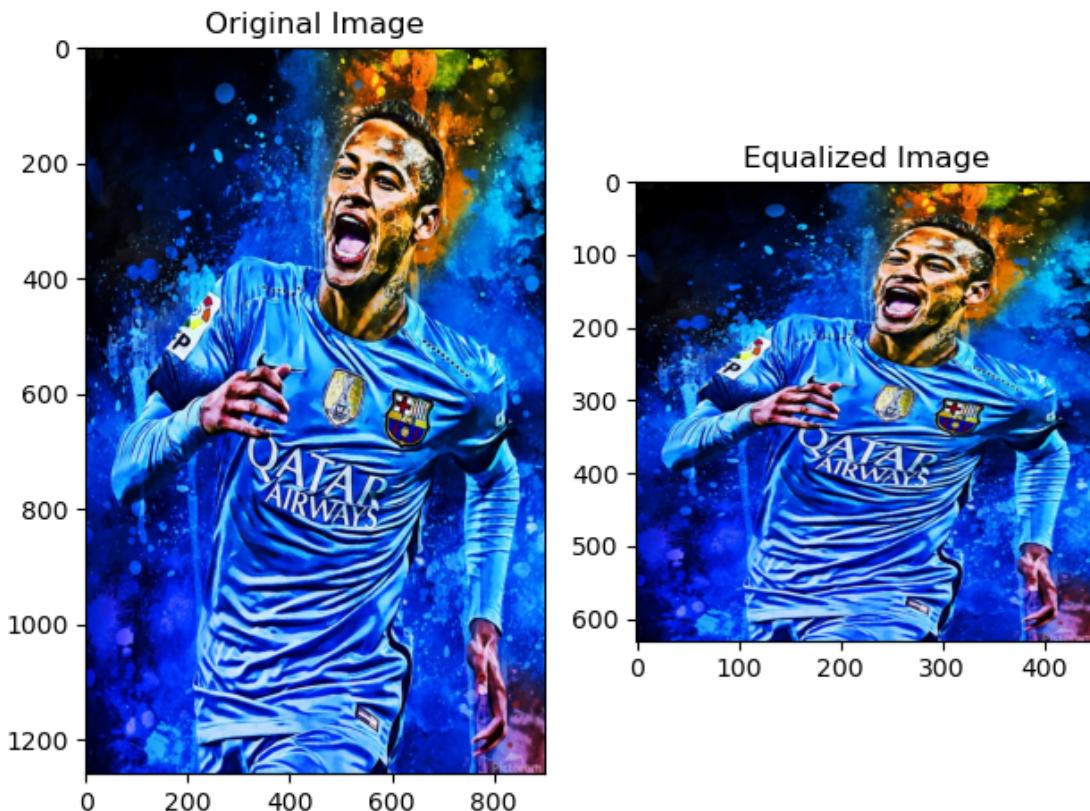
```

```

plt.title('Equalized Image')
plt.gca().set_aspect(original_width/original_height) # Set aspect ratio based
# on original image dimensions

plt.tight_layout()
plt.show()

```



6.4 Lanczos Interpolation

```
[50]: resized_img = original_img.resize((new_width, new_height), resample=Image.
# LANCZOS)
```

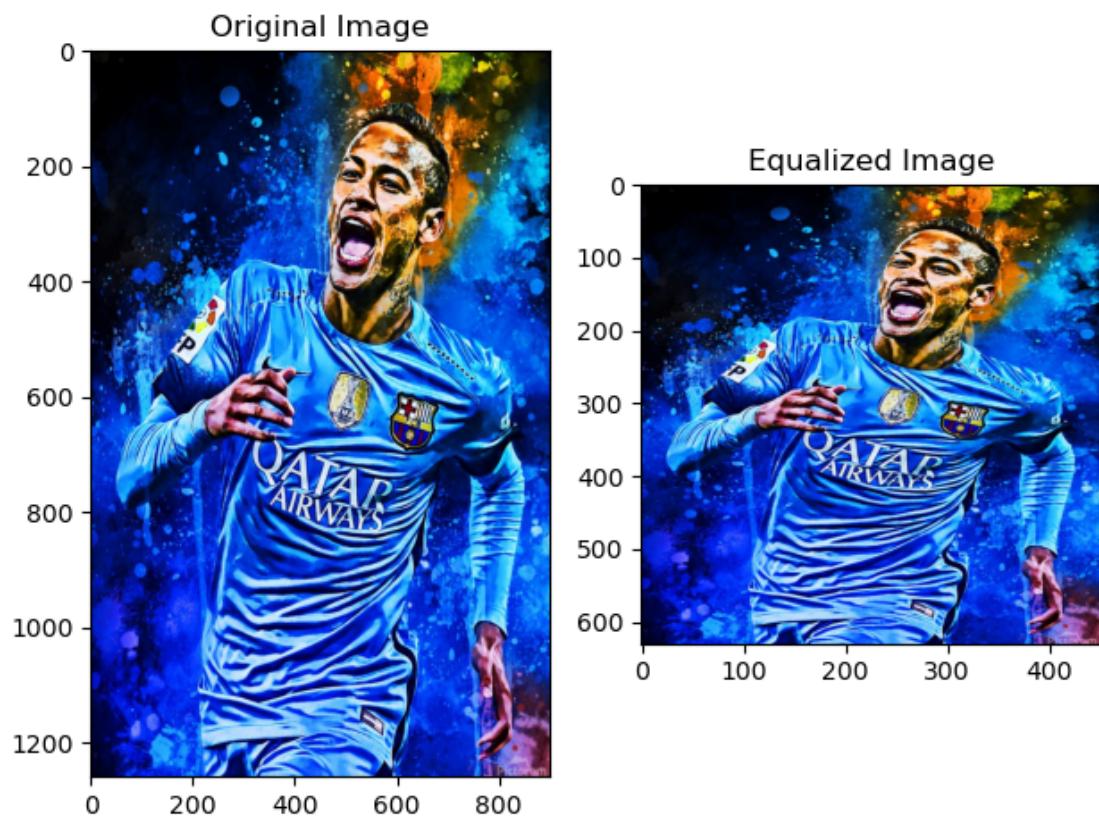
```
[51]: original_width, original_height = original_img.size

plt.subplot(1, 2, 1)
plt.imshow(original_img)
plt.title('Original Image')
plt.gca().set_aspect('auto') # Ensure the aspect ratio is automatic

plt.subplot(1, 2, 2)
```

```
plt.imshow(resized_img)
plt.title('Equalized Image')
plt.gca().set_aspect(original_width/original_height) # Set aspect ratio based
# on original image dimensions

plt.tight_layout()
plt.show()
```



Assignment 16

CUDA programming

1. What is CUDA ?

A. CUDA = Compute Unified Device Architecture

CUDA is a parallel computing platform and programming model developed by NVIDIA for general computing on graphical processing units (GPUs).

It allows to access the raw computing power of CUDA GPUs to process data faster than with traditional CPUs. CUDA can achieve higher parallelism and efficiency than general-purpose CPU code using parallel processes. It enables parallel processing by breaking down a task into thousands of smaller "threads" executed independently.

2. What is the prerequisite for learning CUDA?

- Understanding parallel computing concepts such as **threads**, **blocks**, **grids**, and **synchronization** is crucial for CUDA programming.
- Many applications of CUDA involve numerical computations, such as **matrix operations** and **linear algebra**.
- CUDA enable or **capable GPU** devices, also have to installed **CUDA toolkit** and **nvidia developer driver**.

3. Which are the languages that support CUDA?

A. C++, C, C#, Fortran, Python, Java

4. What do you mean by a CUDA ready architecture?

- A. CUDA Ready Architecture refers to a hardware architecture designed by NVIDIA to support CUDA and their parallel computing platform and programming model. In short the GPU architecture is designed and optimized to support CUDA

5. How CUDA works?

- A. When a processor is given a task, it will pass the instructions for that task to the GPU. The GPU will then do its work, following the instructions from the CPU. GPUs run one

kernel (a group of tasks) at a time. Each kernel consists of blocks, which are independent groups of ALUs. Each block contains threads, which are levels of computation. The threads in each block typically work together to calculate a value. Once the job is completed, the results from the GPU are given back to the CPU.

6. What are the benefits and limitations of CUDA programming?

A. Benefits :

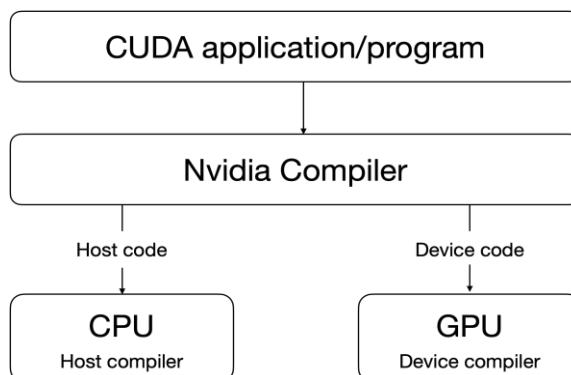
- Massive parallelism
- Performance boost
- Integrated memory and shared memory

Limitations :

- Works only on nvidia GPUs
- Highly Parallelizable Problems Only(Not all problems can be effectively parallelized for GPUs)
- Limited Portability (Porting to other GPU architectures requires significant effort.)

7. Understand and explain the CUDA program structure with an example.

- A. CUDA programming includes code for both the GPU and CPU. By default, a typical C programme is a CUDA programme that simply contains the host code. The CPU is referred to as the host, while the GPU is referred to as the device. While the host code can be compiled using a regular C compiler such as GCC, the device code requires a specific compiler to comprehend the API functions that are used. For Nvidia GPUs, the compiler is known as the NVCC (Nvidia C Compiler).



The GPU runs the device code, whereas the CPU runs the host code. The NVCC runs a CUDA programme and isolates the host code from the device code. To do this, specific CUDA keywords are searched for. The code that is intended to run on the GPU (device code) is identified by special CUDA keywords known as 'Kernels' that

label data-parallel functions. The NVCC further compiles the device code, which is then run on the GPU.

8. Explain CUDA thread organization.

- A. Organizing CUDA threads is similar to forming a team to collaborate on a large project. Here's a simplified breakdown:

Grids: Think of a grid as the overall project. It is made up of several smaller sections known as blocks.

Blocks: Each block represents a smaller team inside the project. They collaborate on a certain aspect of the project and can interact with one another effortlessly.

Threads: Each block has individual workers known as threads. They function similarly to team members who conduct the real work. Each thread completes a little portion of the job allocated to the block.

Divide the work into grids, which are then divided into blocks, with each block containing many threads. This arrangement helps to organize and perform duties efficiently on the GPU.

9. Install and try CUDA sample program and explain the same. (installation steps).

- A. CUDA installation :

Pre-requisite:

- anaconda
- Python
- Tensorflow
- Jupyter
- CUDA enabled graphic card

To install CUDA run this following command in conda power shell or in its environment.

```
conda install -c conda-forge cudatoolkit=11.2 cudnn=8.1.0 -y
```

```

PS C:\Users\Meetrajsinh> conda install -c conda-forge cudatoolkit=11.2 cudnn=8.1.0 -y
Retrieving notices: ...working... done
Collecting package metadata (current_repodata.json): done
Solving environment: unsuccessful initial attempt using frozen solve. Retrying with flexible solve.
Collecting package metadata (repodata.json): done
Solving environment: done

==> WARNING: A newer version of conda exists. <==
  current version: 23.7.4
  latest version: 24.3.0

Please update conda by running

$ conda update -n base -c defaults conda

Or to minimize the number of packages updated during conda update use

  conda install conda=24.3.0

## Package Plan ##

environment location: C:\Users\Meetrajsinh\anaconda3

added / updated specs:
- cudatoolkit=11.2
- cudnn=8.1.0

The following packages will be downloaded:

  package          build
archspec-0.2.3      pyhd8ed1ab_0      48 KB  conda-forge
conda-23.9.0        py311h1ea47a8_2     1.2 MB  conda-forge
cudatoolkit-11.2.2  h93397f7_10     879.9 MB  conda-forge
cudnn-8.1.0.77      h3e0ff4f4_0     610.8 MB  conda-forge
python_abi-3.11      2_cp311           5 KB  conda-forge
truststore-0.8.0    pyhd8ed1ab_0      20 KB  conda-forge

Total:           1.46 GB

```

```

[1]: import tensorflow as tf
[2]: gpus = tf.config.list_physical_devices("GPU")
[6]: if gpus:
    for gpu in gpus:
        print("Found a GPU with the name:", gpu)
    else:
        print("Failed to detect a GPU.")
Found a GPU with the name: PhysicalDevice(name='/physical_device:GPU:0', device_type='GPU')
[7]: !nvidia-smi
Wed Apr  3 15:10:23 2024
+-----+
| NVIDIA-SMI 551.86       Driver Version: 551.86      CUDA Version: 12.4 |
+-----+
| GPU  Name        TCC/WDDM | Bus-Id     Disp.A  Volatile Uncorr. ECC | | | | | | |
| Fan  Temp  Perf  Pwr:Usage/Cap | Memory-Usage | GPU-Util  Compute M. |
| |          |          |             |              |          |          |          |
|-----+
|  0  NVIDIA GeForce RTX 4060 ... WDDM   | 00000000:01:00.0 Off |          N/A |
| N/A   47C   P8    4W / 35W |          0MiB /  8188MiB |  0%   Default |          N/A |
|-----+
+-----+
| Processes:                               GPU Memory |
| GPU  GI  CI PID Type Process name        Usage  |
| ID  ID
|-----+
| No running processes found
+-----+

```

As shown in above image if CUDA was successfully installed then by running above code it will show available gpu and also by running nvidia-smi it will show all detail of gpu and also show CUDA version.

Code example :

```

import numpy as np
from numba import cuda
# Define the CUDA kernel function
@cuda.jit
# @cuda.jit(gridsize=(arr.size // threadsperblock + 1, 1), blocksize=(threadsperblock,))
def add_to_array(arr):
    """
    This kernel adds 1.0 to each element of the input array.
    Each thread processes one element of the array.
    """

    # Get the unique thread index within a block
    idx = cuda.threadIdx.x
    # Check if the thread index is within the array bounds
    if idx < arr.size:
        arr[idx] += 1.0

    # Create a sample NumPy array on the CPU (host)
    arr = np.array([1.0, 2.0, 3.0])
    # Allocate memory for the array on the GPU (device)
    d_arr = cuda.to_device(arr)
    # Configure the number of threads per block (adjust as needed)
    threadsperblock = 512
    gridsize = (arr.size // threadsperblock + 1, 1)
    add_to_array[gridsize, threadsperblock](d_arr)
    # Copy results back from GPU to CPU
    host_arr = d_arr.copy_to_host() # Use copy_to_host instead of to_host
    arr = host_arr # Assign the copied data to the original array
    arr

```

Output :

```
array([3., 4., 5.])
```

Assignment 17

```
[ ]: !nvidia-smi
```

```
Tue Apr 16 07:06:05 2024
+-----+
-----+
| NVIDIA-SMI 535.104.05          Driver Version: 535.104.05    CUDA Version:
12.2      |
|-----+-----+-----+
-----+
| GPU  Name                  Persistence-M | Bus-Id        Disp.A | Volatile
Uncorr. ECC |
| Fan  Temp     Perf          Pwr:Usage/Cap |           Memory-Usage | GPU-Util
Compute M. |
|           |                               |           |
MIG M. |
|=====+=====+=====+=====+=====+
=====|
|   0  Tesla T4                Off  | 00000000:00:04.0 Off |
0 |
| N/A  40C     P8              9W /  70W |     0MiB / 15360MiB |      0%
Default |
|           |           |
N/A |
+-----+-----+-----+
-----+
+-----+
-----+
| Processes:
|
| GPU  GI  CI          PID  Type  Process name             GPU
Memory |
|       ID  ID
Usage  |
|=====+=====+=====+=====+=====+
=====|
```

```
| No running processes found  
|  
+-----  
-----+
```

1.1 1. To print hello message on the screen using kernal function

```
[ ]: %%writefile hello_1_1.cu  
  
#include <stdio.h>  
  
__global__ void cuda_hello_1_1() {  
    printf("Hello World from GPU with grid dimension (1, 1) and block dimension  
    ↴(1, 1)!\\n");  
}  
  
int main() {  
    cuda_hello_1_1<<<1,1>>>();  
    cudaDeviceSynchronize(); // Make sure all GPU work is done before exiting  
    return 0;  
}
```

Writing hello_1_1.cu

```
[ ]: !nvcc -o hello_1_1 hello_1_1.cu
```

```
[ ]: !./hello_1_1
```

Hello World from GPU with grid dimension (1, 1) and block dimension (1, 1)!

1.2 2. To add two vectors of size 100 and 20000 and analyze the performance comparison between cpu and gpu processing

1.2.1 GPU

```
[ ]: !pip install pycuda
```

```
Collecting pycuda  
  Downloading pycuda-2024.1.tar.gz (1.7 MB)  
    1.7/1.7 MB  
12.0 MB/s eta 0:00:00  
  Installing build dependencies ... done  
  Getting requirements to build wheel ... done  
  Preparing metadata (pyproject.toml) ... done  
Collecting pytools>=2011.2 (from pycuda)  
  Downloading pytools-2024.1.1-py2.py3-none-any.whl (85 kB)  
    85.1/85.1 kB  
12.0 MB/s eta 0:00:00  
Requirement already satisfied: appdirs>=1.4.0 in
```

```

/usr/local/lib/python3.10/dist-packages (from pycuda) (1.4.4)
Collecting mako (from pycuda)
  Downloading Mako-1.3.3-py3-none-any.whl (78 kB)
    78.8/78.8 kB
10.9 MB/s eta 0:00:00
Requirement already satisfied: platformdirs>=2.2.0 in
/usr/local/lib/python3.10/dist-packages (from pytools>=2011.2>pycuda) (4.2.0)
Requirement already satisfied: typing-extensions>=4.0 in
/usr/local/lib/python3.10/dist-packages (from pytools>=2011.2>pycuda) (4.11.0)
Requirement already satisfied: MarkupSafe>=0.9.2 in
/usr/local/lib/python3.10/dist-packages (from mako>pycuda) (2.1.5)
Building wheels for collected packages: pycuda
  Building wheel for pycuda (pyproject.toml) ... done
  Created wheel for pycuda: filename=pycuda-2024.1-cp310-cp310-linux_x86_64.whl
size=661204
sha256=51efb7c5582dd86e48b9404a05e0a366352406f4840bf4dc162fe9a89aa2ad1c
  Stored in directory: /root/.cache/pip/wheels/12/34/d2/9a349255a4eca3a486d82c79
d21e138ce2cccd90f414d9d72b8
Successfully built pycuda
Installing collected packages: pytools, mako, pycuda
Successfully installed mako-1.3.3 pycuda-2024.1 pytools-2024.1.1

```

```
[ ]: import numpy as np
import pycuda.driver as cuda
import pycuda.autoinit
from pycuda.compiler import SourceModule
import time
```

```
[ ]: # CUDA kernel function to add two vectors
cuda_kernel_code = """
__global__ void vector_add(float *a, float *b, float *c, int n) {
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    if (i < n) {
        c[i] = a[i] + b[i];
    }
}
"""
```

```
[ ]: cuda_module = SourceModule(cuda_kernel_code)
vector_add_cuda = cuda_module.get_function("vector_add")
```

```
[ ]: def vector_add_gpu(a, b):
    n = a.size

    a_gpu = cuda.mem_alloc(a.nbytes)
    b_gpu = cuda.mem_alloc(b.nbytes)
    c_gpu = cuda.mem_alloc(b.nbytes)
```

```

    cuda.memcpy_htod(a_gpu, a)
    cuda.memcpy_htod(b_gpu, b)

    block_dim = (256, 1, 1)
    grid_dim = ((n + block_dim[0] - 1) // block_dim[0], 1)

    start_time = time.time()

    vector_add_cuda(a_gpu, b_gpu, c_gpu, np.int32(n), block=block_dim, u
    ↵grid=grid_dim)

    cuda.Context.synchronize()

    end_time = time.time()

    c = np.empty_like(a)
    cuda.memcpy_dtoh(c, c_gpu)

    return c, end_time - start_time

```

```

[ ]: vector_size_1 = 100
vector_size_2 = 20000
a = np.random.randn(vector_size_2).astype(np.float32)
b = np.random.randn(vector_size_2).astype(np.float32)

result_gpu1, gpu_time1 = vector_add_gpu(a[:vector_size_1], b[:vector_size_1])
result_gpu2, gpu_time2 = vector_add_gpu(a[:vector_size_2], b[:vector_size_2])

```

```

[ ]: print("Vector addition of size", vector_size_1, "on GPU took", gpu_time1, u
      ↵"seconds.")

print("Vector addition of size", vector_size_2, "on GPU took", gpu_time2, u
      ↵"seconds.")

```

Vector addition of size 100 on GPU took 0.0007643699645996094 seconds.
 Vector addition of size 20000 on GPU took 6.818771362304688e-05 seconds.

1.2.2 CPU

```
[1]: import numpy as np
import time
```

```
[10]: def vector_add_cpu(a, b):
        start_time = time.time()
        result = a + b
        end_time = time.time()
        return result, end_time - start_time
```

```
[11]: vector_size_1 = 100
vector_size_2 = 20000
a = np.random.randn(vector_size_2).astype(np.float32)
b = np.random.randn(vector_size_2).astype(np.float32)

result_cpu1, cpu_time1 = vector_add_cpu(a[:vector_size_1], b[:vector_size_1])
result_cpu2, cpu_time2 = vector_add_cpu(a[:vector_size_2], b[:vector_size_2])
```

```
[12]: print("Vector addition of size", vector_size_1, "on CPU took", cpu_time1, "seconds.")
print("Vector addition of size", vector_size_2, "on CPU took", cpu_time2, "seconds.")
```

Vector addition of size 100 on CPU took 0.0 seconds.

Vector addition of size 20000 on CPU took 0.0 seconds.

- Vector addition of size 100 on GPU took 0.0007691383361816406 seconds.
- Vector addition of size 20000 on GPU took 7.128715515136719e-05 seconds.

1.3 3. To multiply two matrix of size 20 X 20 and 1024 X 1024 analyze the performance comparison between cpu and gpu processing

1.3.1 GPU

```
[ ]: def matrix_multiply_gpu(a, b):
    cuda_code = """
        __global__ void matrix_multiply(float *a, float *b, float *c, int n) {
            int row = blockIdx.y * blockDim.y + threadIdx.y;
            int col = blockIdx.x * blockDim.x + threadIdx.x;

            if (row < n && col < n) {
                float sum = 0.0;
                for (int i = 0; i < n; ++i) {
                    sum += a[row * n + i] * b[i * n + col];
                }
                c[row * n + col] = sum;
            }
        }
    """

    mod = SourceModule(cuda_code)

    matrix_multiply_cuda = mod.get_function("matrix_multiply")

    a_gpu = cuda.mem_alloc(a.nbytes)
    b_gpu = cuda.mem_alloc(b.nbytes)
    c_gpu = cuda.mem_alloc(a.nbytes)

    cuda.memcpy_htod(a_gpu, a)
```

```

        cuda.memcpy_htod(b_gpu, b)

        block_size = (16, 16, 1)
        grid_size = ((a.shape[1] + block_size[0] - 1) // block_size[0], (a.shape[0] +
        ↵+ block_size[1] - 1) // block_size[1], 1)

        matrix_multiply_cuda(a_gpu, b_gpu, c_gpu, np.int32(a.shape[0]), ↵
        ↵block=block_size, grid=grid_size)

        c = np.empty_like(a)
        cuda.memcpy_dtoh(c, c_gpu)

    return c

```

[]: def generate_random_matrix(rows, cols):
 return np.random.rand(rows, cols).astype(np.float32)

[]: def measure_time(matrix_size, func, *args):
 start_time = time.time()
 result = func(*args)
 end_time = time.time()
 return result, end_time - start_time

[]: matrix_sizes = [(20, 20), (1024, 1024)]

[]: for size in matrix_sizes:
 print(f"\nMatrix size: {size}")
 a = generate_random_matrix(*size)
 b = generate_random_matrix(*size)

 gpu_result, gpu_time = measure_time(size, matrix_multiply_gpu, a, b)
 print(f"GPU time: {gpu_time:.6f} seconds")

Matrix size: (20, 20)
GPU time: 0.428407 seconds

Matrix size: (1024, 1024)
GPU time: 0.018636 seconds

1.3.2 CPU

[2]: def matrix_multiply_cpu(a, b):
 result = np.zeros((a.shape[0], b.shape[1]), dtype=np.float32)
 for i in range(a.shape[0]):
 for j in range(b.shape[1]):
 for k in range(a.shape[1]):
 result[i, j] += a[i, k] * b[k, j]

```

    return result

[3]: def generate_random_matrix(rows, cols):
    return np.random.rand(rows, cols).astype(np.float32)

[4]: def measure_time(matrix_size, func, *args):
    start_time = time.time()
    result = func(*args)
    end_time = time.time()
    return result, end_time - start_time

[5]: matrix_sizes = [(20, 20), (1024, 1024)]

[6]: for size in matrix_sizes:
    print(f"\nMatrix size: {size}")
    a = generate_random_matrix(*size)
    b = generate_random_matrix(*size)

    # CPU matrix multiplication
    cpu_result, cpu_time = measure_time(size, matrix_multiply_cpu, a, b)
    print(f"CPU time: {cpu_time:.6f} seconds")

```

Matrix size: (20, 20)
CPU time: 0.000000 seconds

Matrix size: (1024, 1024)
CPU time: 533.798448 seconds

- Matrix size: (20, 20)
- GPU time: 0.703994 seconds
- Matrix size: (1024, 1024)
- GPU time: 0.014648 seconds

1.4 4. To obtain CUDA device information and print the output

```

[ ]: import pycuda.driver as cuda

# Initialize PyCUDA
cuda.init()

num_devices = cuda.Device.count()

print("Number of CUDA devices:", num_devices)

for i in range(num_devices):
    device = cuda.Device(i)

```

```
print("\nCUDA Device:", i)
print("  Name:", device.name())
print("  Compute Capability:", device.compute_capability())
print("  Total Memory:", device.total_memory() / (1024 ** 3), "GB")
print("  Max Threads per Block:", device.max_threads_per_block)
print("  Multiprocessor Count:", device.multiprocessor_count)
print("  Clock Rate:", device.clock_rate / 1e6, "GHz")
```

Number of CUDA devices: 1

```
CUDA Device: 0
Name: Tesla T4
Compute Capability: (7, 5)
Total Memory: 14.74810791015625 GB
Max Threads per Block: 1024
Multiprocessor Count: 40
Clock Rate: 1.59 GHz
```

Assignment 18

1.1 Image Processing CUDA

1.2 Implement the following Image Processing operations in sequential and parallel using CUDA Programming.

1.2.1 Gaussian blur

Describe Gaussian Blur in brief. Gaussian blur is a popular image processing technique used to minimize visual noise and detail, resulting in a smoother look. Gaussian blur convolves each pixel in the image with a Gaussian kernel, a 2D matrix that represents a bell-shaped curve. The size and standard deviation (sigma) of the Gaussian kernel control the amount of blurring applied to the image. A larger kernel size and a higher sigma value cause more noticeable blurring. During convolution, each pixel in the image is replaced by a weighted average of its neighbors, the weights of which are defined by the Gaussian function. This method efficiently decreases the image's high-frequency components, smoothing out sharp transitions and noise.

Where parallelism can be inserted? Parallelism can be inserted in Gaussian blur at key stages:

1. image Copy and Memory Allocation: Transfer input picture data to GPU while allocating memory for the output image in parallel.
2. Convolution: Perform the convolution process with CUDA kernels, with each thread computing the weighted sum for a given output pixel.
3. Border Handling: Process border regions concurrently, allocating various threads to handle different parts of the border.
4. image Copy Back: Transfer processed picture data from the GPU to the host RAM in parallel.

Analyze the performance in serial and parallel model.

Serial

```
[28]: import os  
import cv2  
import numpy as np  
import time
```

```
[29] :
```

```

def gaussian_kernel(size, sigma=1):
    kernel = np.fromfunction(lambda x, y: (1/(2*np.pi*sigma**2)) * np.
    ↪exp(-((x-(size-1)/2)**2 + (y-(size-1)/2)**2)/(2*sigma**2)), (size, size))
    return kernel / np.sum(kernel)

```

```
[30]: def gaussian_blur_sequential(image, kernel_size):
    kernel = gaussian_kernel(kernel_size)
    blurred_image = cv2.filter2D(image, -1, kernel) # Using OpenCV's filter2D
    ↪for convolution
    return blurred_image

```

```
[31]: def process_images_in_folder(folder_path, kernel_size):
    times = {}
    for filename in os.listdir(folder_path):
        if filename.endswith(".jpg") or filename.endswith(".png"):
            image_path = os.path.join(folder_path, filename)
            image = cv2.imread(image_path, cv2.IMREAD_COLOR)
            start_time = time.time()
            blurred_image = gaussian_blur_sequential(image, kernel_size)
            end_time = time.time()
            times[filename] = end_time - start_time
    return times

```

```
[32]: folder_path = "/content/drive/MyDrive/data_set/minion"
kernel_size = 3

```

```
[33]: start_time_total = time.time()
times_per_image = process_images_in_folder(folder_path, kernel_size)
end_time_total = time.time()
total_time = end_time_total - start_time_total

```

```
[34]: print("Time taken for Gaussian blur on each image:")
for filename, time_taken in times_per_image.items():
    print(f"{filename}: {time_taken:.4f} seconds")

```

Time taken for Gaussian blur on each image:

```

1.jpg: 0.0348 seconds
2.jpg: 0.0024 seconds
3.jpg: 0.0618 seconds
4.jpg: 0.0038 seconds
5.jpg: 0.0173 seconds
6.jpg: 0.0064 seconds
7.jpg: 0.0085 seconds
8.jpg: 0.0044 seconds
9.jpg: 0.0034 seconds
10.jpg: 0.0033 seconds
11.jpg: 0.0027 seconds

```

```
[35]: print(f"\nTotal time taken for all images: {total_time:.4f} seconds")
```

Total time taken for all images: 0.6478 seconds

Parallel

```
[36]: import copy as cp
from PIL import Image
import os
import time

def process_image(image_array):
    def gaussian_kernel(size, sigma=1):
        kernel_1D = cp.linspace(-(size // 2), size // 2, size)
        for i in range(size):
            kernel_1D[i] = cp.exp(-0.5 * (kernel_1D[i] / sigma) ** 2)
        kernel_2D = cp.outer(kernel_1D, kernel_1D)
        kernel_2D /= kernel_2D.sum()
        return kernel_2D

    kernel_size = 5
    gaussian_kernel_array = gaussian_kernel(kernel_size)
    blurred_image = cp.asarray(Image.fromarray(cp.asnumpy(image_array)).
        convert("L"))

    return blurred_image

directory = "/content/drive/MyDrive/data_set/minion"
num_images = 0
start_time = time.time()

image_paths = [os.path.join(directory, filename) for filename in os.
    listdir(directory) if filename.endswith(".jpg")]
image_arrays = [cp.array(Image.open(image_path).convert("L")) for image_path in_
    image_paths]
processed_images = [process_image(image_array) for image_array in image_arrays]

cp.cuda.Device().synchronize()

total_time_parallel = time.time() - start_time
num_images = len(image_paths)

print("Number of images processed in parallel:", num_images)
print("Time taken for parallel processing:", total_time_parallel, "seconds")
```

Number of images processed in parallel: 11

Time taken for parallel processing: 0.41269755363464355 seconds

1.2.2 FFT- Fast Fourier Transform

Describe FFT in brief The FFT algorithm decomposes the DFT computation into smaller subproblems, exploiting the periodicity and symmetry properties of complex exponential functions. By iteratively applying these subproblems and integrating the answers, the FFT computes the DFT in much less operations than direct computation.

The FFT output represents the input signal's frequency spectrum, including the magnitude and phase of each frequency component. This information is useful for filtering, noise reduction, spectrum analysis, and feature extraction.

Where parallelism can be inserted? Parallelism in the Fast Fourier Transform (FFT) can be introduced at numerous stages:

1. Data Splitting: Split incoming data into chunks that can be processed independently by several cores or threads
2. Butterfly Operations: Parallelize butterfly operations, particularly on SIMD architectures such as GPUs.
3. Twiddle Factor Calculation: Calculate twiddle factors simultaneously across several processor units.
4. Memory Access: Improve memory access patterns for parallel processing, such as coalesced memory access and prefetching.
5. Inverse FFT (IFFT): Use parallelization techniques to improve inverse FFT computation performance.

Analyze the performance in serial and parallel model.

serial

```
[37]: def fft_sequential(image):
    fft_image = np.fft.fft2(image)
    return fft_image
```

```
[38]: def process_images_in_folder_fft(folder_path):
    times = []
    for filename in os.listdir(folder_path):
        if filename.endswith(".jpg") or filename.endswith(".png"):
            image_path = os.path.join(folder_path, filename)
            image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
            start_time = time.time()
            fft_image = fft_sequential(image)
            end_time = time.time()
            times[filename] = end_time - start_time
    return times
```

```
[39]: folder_path = "/content/drive/MyDrive/data_set/minion"
```

```
[40]: start_time_total = time.time()
times_per_image_fft = process_images_in_folder_fft(folder_path)
end_time_total = time.time()
total_time_fft = end_time_total - start_time_total
```

```
[41]: print("Time taken for FFT on each image:")
for filename, time_taken in times_per_image_fft.items():
    print(f"{filename}: {time_taken:.4f} seconds")
```

Time taken for FFT on each image:

```
1.jpg: 0.0262 seconds
2.jpg: 0.0302 seconds
3.jpg: 1.3454 seconds
4.jpg: 0.0321 seconds
5.jpg: 0.2315 seconds
6.jpg: 0.0642 seconds
7.jpg: 0.0505 seconds
8.jpg: 0.0690 seconds
9.jpg: 0.0231 seconds
10.jpg: 0.0405 seconds
11.jpg: 0.0352 seconds
```

```
[42]: print(f"\nTotal time taken for FFT on all images: {total_time_fft:.4f} seconds")
```

Total time taken for FFT on all images: 2.2286 seconds

Parallel

```
[43]: import os
import time
import cupy as cp
from PIL import Image

directory = "/content/drive/MyDrive/data_set/minion"

num_images = 0
start_time = time.time()

def process_image(image_array):
    global num_images
    num_images += 1

    fft_image = cp.fft.fft2(image_array)
    fft_image_shifted = cp.fft.fftshift(fft_image)

    rows, cols = image_array.shape
    center_row, center_col = rows // 2, cols // 2
    radius = 20
```

```

high_pass_filter = cp.ones((rows, cols))
mask = cp.zeros((rows, cols))
mask[center_row - radius:center_row + radius, center_col - radius:
    ↵center_col + radius] = 1
high_pass_filter -= mask

filtered_image_fft = fft_image_shifted * high_pass_filter

filtered_image = cp.abs(cp.fft.ifft2(cp.fft.ifftshift(filtered_image_fft)))
return filtered_image

image_paths = [os.path.join(directory, filename) for filename in os.
    ↵listdir(directory) if filename.endswith(".jpg")]
image_arrays = [cp.array(Image.open(image_path).convert("L")) for image_path in
    ↵image_paths]
processed_images = [process_image(image_array) for image_array in image_arrays]

cp.cuda.Device().synchronize()

total_time_parallel = time.time() - start_time

print("Number of images processed in parallel:", num_images)
print("Time taken for parallel processing:", total_time_parallel, "seconds")

```

Number of images processed in parallel: 11
Time taken for parallel processing: 0.7189762592315674 seconds

Assignment 19

Final Learning Synopsis Submission

Some of the practicals focuses on parallel programming paradigms like OpenMP, MPI, and CUDA, as well as image processing and system analysis. Each practical, from matrix multiplication and PI calculation with OpenMP to implementing the Producer-Consumer problem with Python threading, demands coding, analysis, and, in many cases, visualization. The tasks investigate parallelism efficiency, scaling behavior, and the performance differences between sequential and parallel processing. There is also an emphasis on understanding MPI communication, CUDA programming, and other image processing processes. Overall, it's a thorough learning experience targeted at mastering parallel programming concepts and their applications of high performance computing.