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.

Using Numpy

In [1]:
```
import time
import numpy as np
import concurrent.futures
import pandas as pd

def sequential_matrix_multiply(matrix_a, matrix_b):
    return np.dot(matrix_a, matrix_b)

def parallel_matrix_multiply(matrix_a, matrix_b, num_threads):
    with concurrent.futures.ThreadPoolExecutor(max_workers=num_threads) as executor:
        result = np.zeros_like(matrix_a)
        chunk_size = len(matrix_a) // num_threads
        futures = []

        for i in range(num_threads):
            start_idx = i * chunk_size
            end_idx = start_idx + chunk_size
            futures.append(executor.submit(np.dot, matrix_a[start_idx:end_idx], matrix_b, o

        concurrent.futures.wait(futures)

    return result

matrix_sizes = [(5, 5), (50, 50), (100, 100), (250, 250),(500,500), (750, 750), (1000, 1000

results_list = []

for matrix_size in matrix_sizes:
    rows, cols = matrix_size
    matrix_a = np.random.rand(rows, cols)
    matrix_b = np.random.rand(cols, rows)

    start_time = time.time()
    result_seq = sequential_matrix_multiply(matrix_a, matrix_b)
    sequential_time = time.time() - start_time

    for num_threads in [1,2, 4, 8, 10, 15, 20]:
        start_time = time.time()
        result_parallel = parallel_matrix_multiply(matrix_a, matrix_b, num_threads)
        parallel_time = time.time() - start_time

        results_list.append({
            'Matrix Size': matrix_size,
            'Threads': num_threads,
            'Sequential Time': sequential_time,
            'Parallel Time': parallel_time
        })

df = pd.DataFrame(results_list)
print(df)
```

	Matrix Size	Threads	Sequential Time	Parallel Time
0	(5, 5)	1	0.011003	0.000802
1	(5, 5)	2	0.011003	0.000000
2	(5, 5)	4	0.011003	0.004509
3	(5, 5)	8	0.011003	0.000000
4	(5, 5)	10	0.011003	0.000000
5	(5, 5)	15	0.011003	0.000000
6	(5, 5)	20	0.011003	0.011161
7	(50, 50)	1	0.000218	0.000912
8	(50, 50)	2	0.000218	0.001208
9	(50, 50)	4	0.000218	0.001598
10	(50, 50)	8	0.000218	0.000000
11	(50, 50)	10	0.000218	0.000000
12	(50, 50)	15	0.000218	0.008742
13	(50, 50)	20	0.000218	0.003911
14	(100, 100)	1	0.001625	0.000000
15	(100, 100)	2	0.001625	0.000000
16	(100, 100)	4	0.001625	0.005448
17	(100, 100)	8	0.001625	0.001705
18	(100, 100)	10	0.001625	0.002549
19	(100, 100)	15	0.001625	0.004352
20	(100, 100)	20	0.001625	0.005567
21	(250, 250)	1	0.000000	0.004160
22	(250, 250)	2	0.000000	0.000000
23	(250, 250)	4	0.000000	0.004624
24	(250, 250)	8	0.000000	0.001032
25	(250, 250)	10	0.000000	0.000000
26	(250, 250)	15	0.000000	0.009073
27	(250, 250)	20	0.000000	0.008928
28	(500, 500)	1	0.016888	0.000000
29	(500, 500)	2	0.016888	0.016453
30	(500, 500)	4	0.016888	0.000000
31	(500, 500)	8	0.016888	0.014294
32	(500, 500)	10	0.016888	0.017571
33	(500, 500)	15	0.016888	0.016688
34	(500, 500)	20	0.016888	0.019125
35	(750, 750)	1	0.016989	0.018687
36	(750, 750)	2	0.016989	0.022025
37	(750, 750)	4	0.016989	0.024292
38	(750, 750)	8	0.016989	0.032498
39	(750, 750)	10	0.016989	0.030512
40	(750, 750)	15	0.016989	0.042504
41	(750, 750)	20	0.016989	0.046666
42	(1000, 1000)	1	0.039452	0.033872
43	(1000, 1000)	2	0.039452	0.033255
44	(1000, 1000)	4	0.039452	0.049953
45	(1000, 1000)	8	0.039452	0.049914
46	(1000, 1000)	10	0.039452	0.056389
47	(1000, 1000)	15	0.039452	0.060067
48	(1000, 1000)	20	0.039452	0.091283
49	(2000, 2000)	1	0.236901	0.239623
50	(2000, 2000)	2	0.236901	0.253801
51	(2000, 2000)	4	0.236901	0.259886
52	(2000, 2000)	8	0.236901	0.299354
53	(2000, 2000)	10	0.236901	0.326929
54	(2000, 2000)	15	0.236901	0.298494
55	(2000, 2000)	20	0.236901	0.483097

Using loop

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

In [3]:
```
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
```

In [4]:
```
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))

    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)
```

In [5]:
```
def main():
    matrix_sizes = [5, 50, 100, 250, 500, 750, 1000, 2000]
    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)
```

In [6]:
```
if __name__ == "__main__":
    main()
```

In [7]:
```
df2 = pd.read_csv('matrix_multiplication_results.csv')
df2
```

Out[7]:

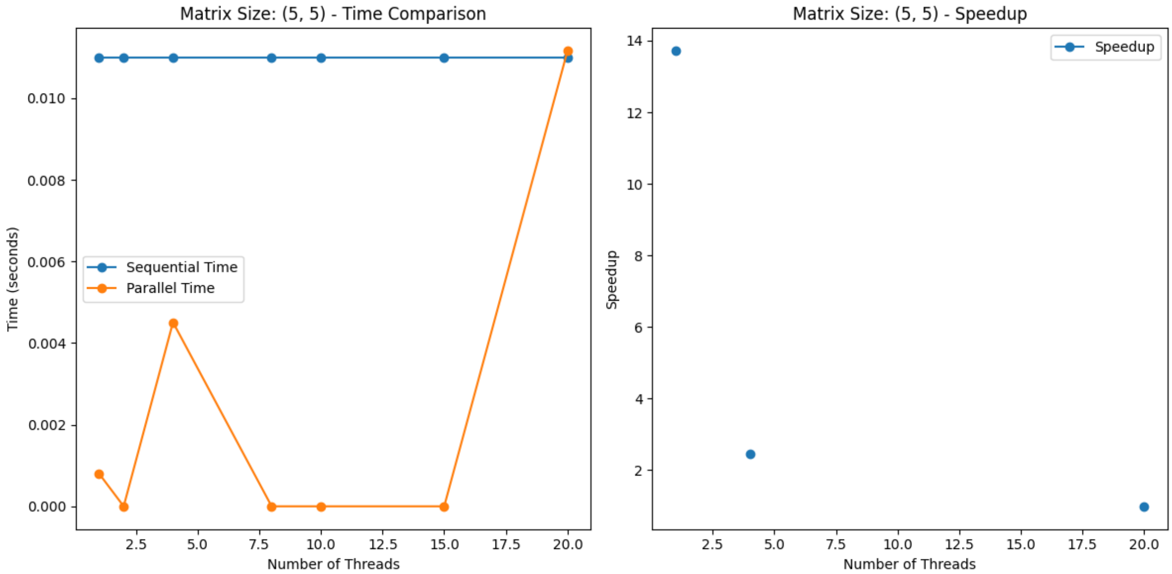
	Matrix Size	Threads	Serial Time	Parallel Time	Speedup	Efficiency
0	5	1	1.000000e-10	1.000000e-10	1.000000e+00	1.000000e+00
1	5	2	1.000000e-10	4.018307e-03	2.488610e-08	1.244305e-08
2	5	4	1.000000e-10	1.000000e-10	1.000000e+00	2.500000e-01
3	5	8	1.000000e-10	5.263329e-03	1.899938e-08	2.374923e-09
4	5	10	1.000000e-10	1.527548e-03	6.546440e-08	6.546440e-09
5	5	15	1.000000e-10	2.702475e-03	3.700312e-08	2.466875e-09
6	5	20	1.000000e-10	2.448320e-03	4.084433e-08	2.042216e-09
7	50	1	5.102158e-04	1.000000e-10	5.102158e+06	5.102158e+06
8	50	2	5.102158e-04	1.000000e-10	5.102158e+06	2.551079e+06
9	50	4	5.102158e-04	2.510309e-03	2.032482e-01	5.081204e-02
10	50	8	5.102158e-04	3.278017e-03	1.556477e-01	1.945596e-02
11	50	10	5.102158e-04	3.556013e-03	1.434797e-01	1.434797e-02
12	50	15	5.102158e-04	7.309675e-03	6.980006e-02	4.653337e-03
13	50	20	5.102158e-04	8.510828e-03	5.994902e-02	2.997451e-03
14	100	1	1.819134e-04	1.000000e-10	1.819134e+06	1.819134e+06
15	100	2	1.819134e-04	1.000000e-10	1.819134e+06	9.095669e+05
16	100	4	1.819134e-04	1.000000e-10	1.819134e+06	4.547834e+05
17	100	8	1.819134e-04	4.508734e-03	4.034689e-02	5.043361e-03
18	100	10	1.819134e-04	1.000000e-10	1.819134e+06	1.819134e+05
19	100	15	1.819134e-04	1.112342e-02	1.635409e-02	1.090273e-03
20	100	20	1.819134e-04	4.046917e-03	4.495110e-02	2.247555e-03
21	250	1	1.000000e-10	1.000000e-10	1.000000e+00	1.000000e+00
22	250	2	1.000000e-10	3.465891e-03	2.885261e-08	1.442631e-08
23	250	4	1.000000e-10	1.788378e-03	5.591660e-08	1.397915e-08
24	250	8	1.000000e-10	1.000000e-10	1.000000e+00	1.250000e-01
25	250	10	1.000000e-10	1.024652e-02	9.759416e-09	9.759416e-10
26	250	15	1.000000e-10	5.845308e-03	1.710774e-08	1.140516e-09
27	250	20	1.000000e-10	6.284475e-03	1.591223e-08	7.956114e-10
28	500	1	1.000000e-10	1.000000e-10	1.000000e+00	1.000000e+00
29	500	2	1.000000e-10	1.000000e-10	1.000000e+00	5.000000e-01
30	500	4	1.000000e-10	1.000000e-10	1.000000e+00	2.500000e-01
31	500	8	1.000000e-10	1.815844e-02	5.507082e-09	6.883853e-10
32	500	10	1.000000e-10	1.213121e-02	8.243198e-09	8.243198e-10
33	500	15	1.000000e-10	1.550841e-02	6.448113e-09	4.298742e-10
34	500	20	1.000000e-10	2.527094e-02	3.957115e-09	1.978557e-10
35	750	1	1.000000e-10	1.000000e-10	1.000000e+00	1.000000e+00
36	750	2	1.000000e-10	1.000000e-10	1.000000e+00	5.000000e-01
37	750	4	1.000000e-10	1.000000e-10	1.000000e+00	2.500000e-01
38	750	8	1.000000e-10	1.566434e-02	6.383927e-09	7.979909e-10
39	750	10	1.000000e-10	1.181316e-02	8.465133e-09	8.465133e-10
40	750	15	1.000000e-10	1.000000e-10	1.000000e+00	6.666667e-02
41	750	20	1.000000e-10	1.000000e-10	1.000000e+00	5.000000e-02
42	1000	1	1.000000e-10	1.000000e-10	1.000000e+00	1.000000e+00
43	1000	2	1.000000e-10	1.000000e-10	1.000000e+00	5.000000e-01
44	1000	4	1.000000e-10	1.000000e-10	1.000000e+00	2.500000e-01

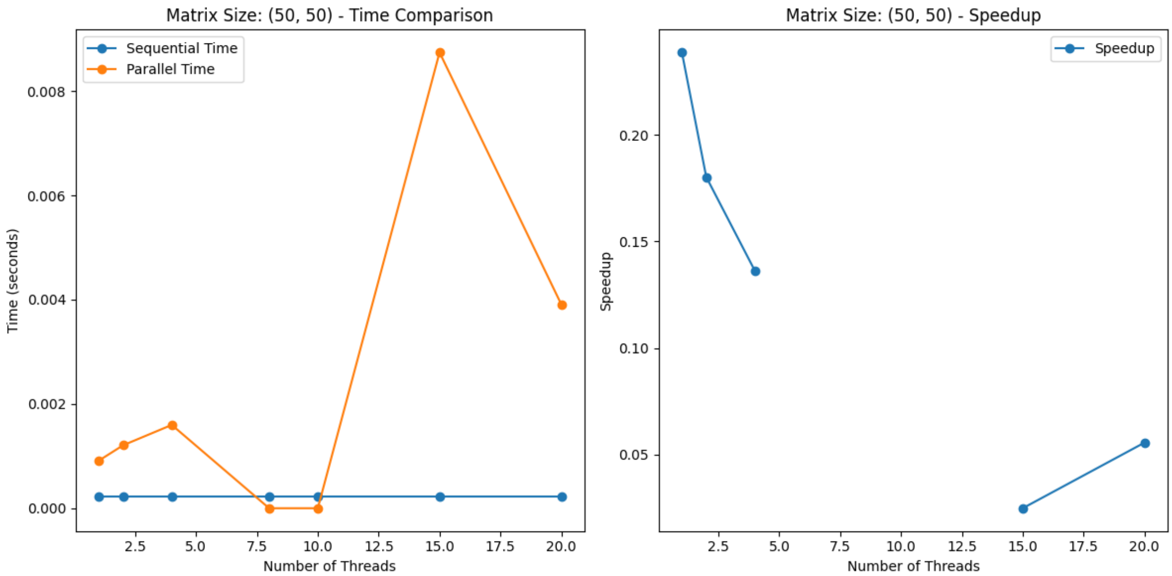
	Matrix Size	Threads	Serial Time	Parallel Time	Speedup	Efficiency
45	1000	8	1.000000e-10	1.721144e-02	5.810090e-09	7.262613e-10
46	1000	10	1.000000e-10	1.010704e-02	9.894093e-09	9.894093e-10
47	1000	15	1.000000e-10	6.479263e-03	1.543385e-08	1.028924e-09
48	1000	20	1.000000e-10	1.561546e-02	6.403909e-09	3.201954e-10
49	2000	1	1.000000e-10	1.000000e-10	1.000000e+00	1.000000e+00
50	2000	2	1.000000e-10	1.000000e-10	1.000000e+00	5.000000e-01
51	2000	4	1.000000e-10	5.483627e-04	1.823610e-07	4.559026e-08
52	2000	8	1.000000e-10	1.000000e-10	1.000000e+00	1.250000e-01
53	2000	10	1.000000e-10	3.440619e-03	2.906454e-08	2.906454e-09
54	2000	15	1.000000e-10	1.400876e-02	7.138390e-09	4.758927e-10
55	2000	20	1.000000e-10	1.000000e-10	1.000000e+00	5.000000e-02

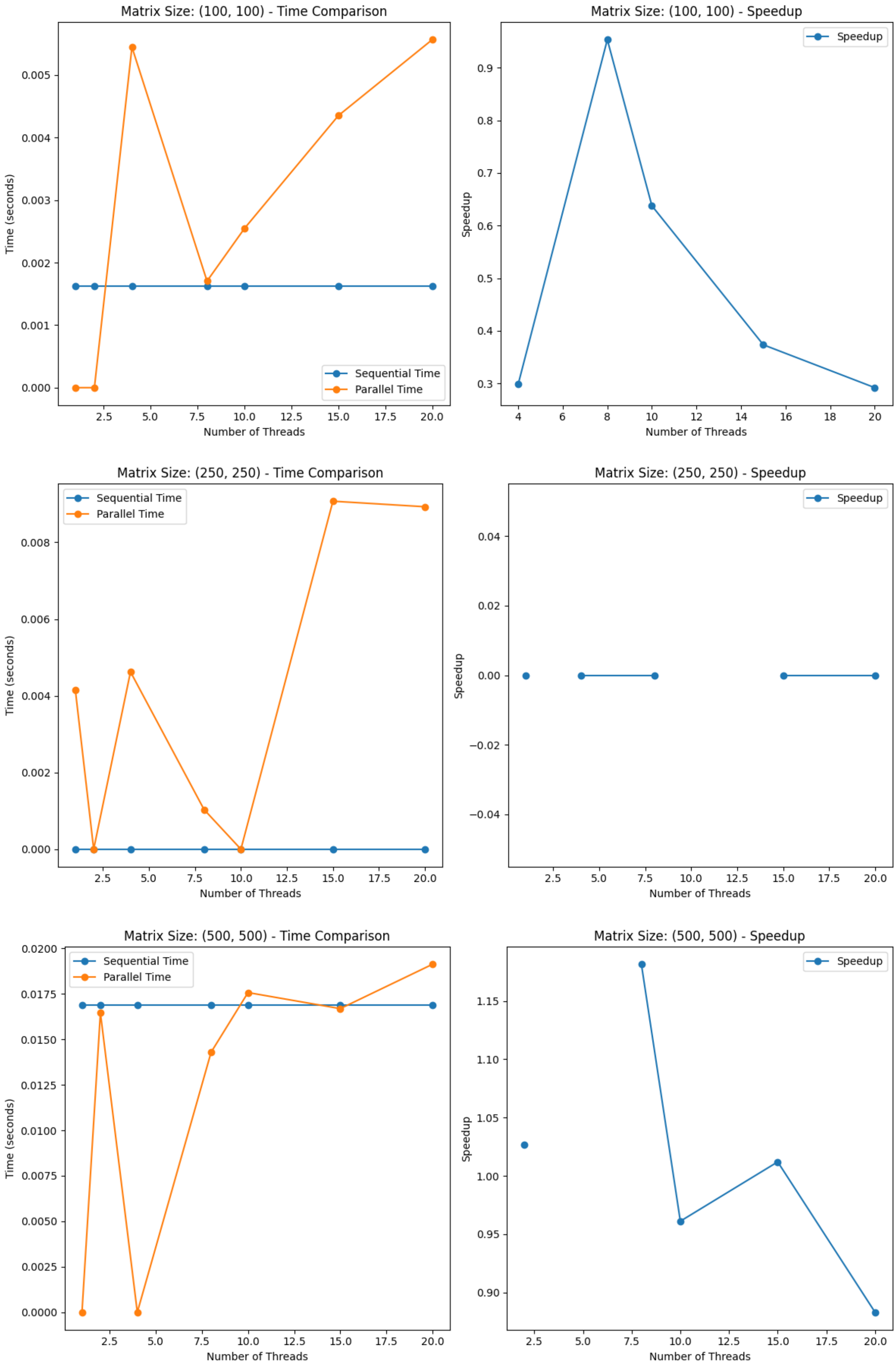
Display a visualization of performance comparison between serial, parallel and Numpy code

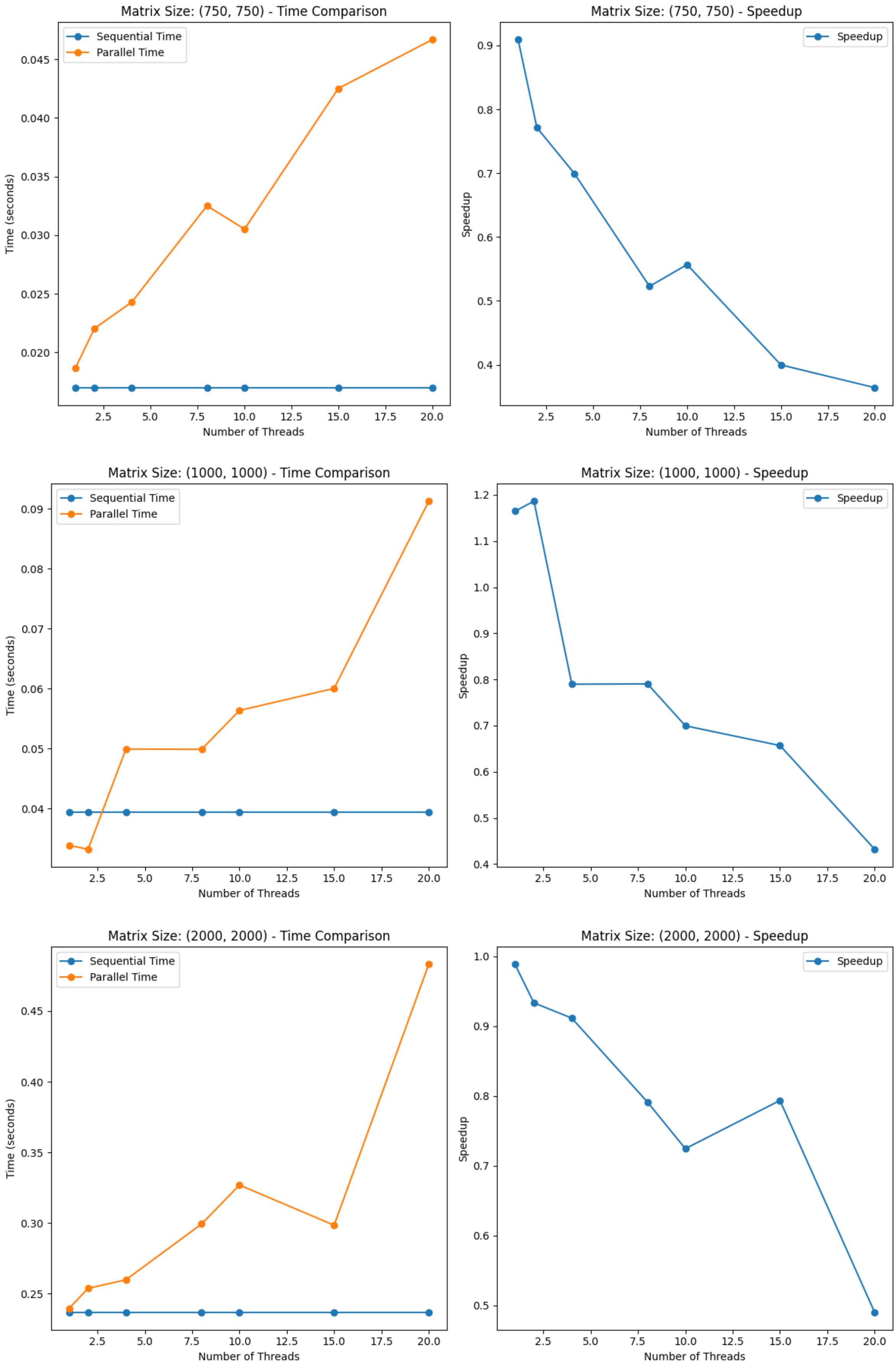
visualization of performance comparison between serial, parallel using Numpy code

```python
In [8]: import matplotlib.pyplot as plt
matrix_sizes = df['Matrix Size'].unique()

for matrix_size in matrix_sizes:
    subset_df = df[df['Matrix Size'] == matrix_size]

    plt.figure(figsize=(12, 6))

    plt.subplot(1, 2, 1)
    plt.plot(subset_df['Threads'], subset_df['Sequential Time'], marker='o', label='Sequent
    plt.plot(subset_df['Threads'], subset_df['Parallel Time'], marker='o', label='Parallel
    plt.title(f"Matrix Size: {matrix_size} - Time Comparison")
    plt.xlabel("Number of Threads")
    plt.ylabel("Time (seconds)")
    plt.legend()

    plt.subplot(1, 2, 2)
    speedup = subset_df['Sequential Time'] / subset_df['Parallel Time']
    plt.plot(subset_df['Threads'], speedup, marker='o', label='Speedup')
    plt.title(f"Matrix Size: {matrix_size} - Speedup")
    plt.xlabel("Number of Threads")
    plt.ylabel("Speedup")
    plt.legend()

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



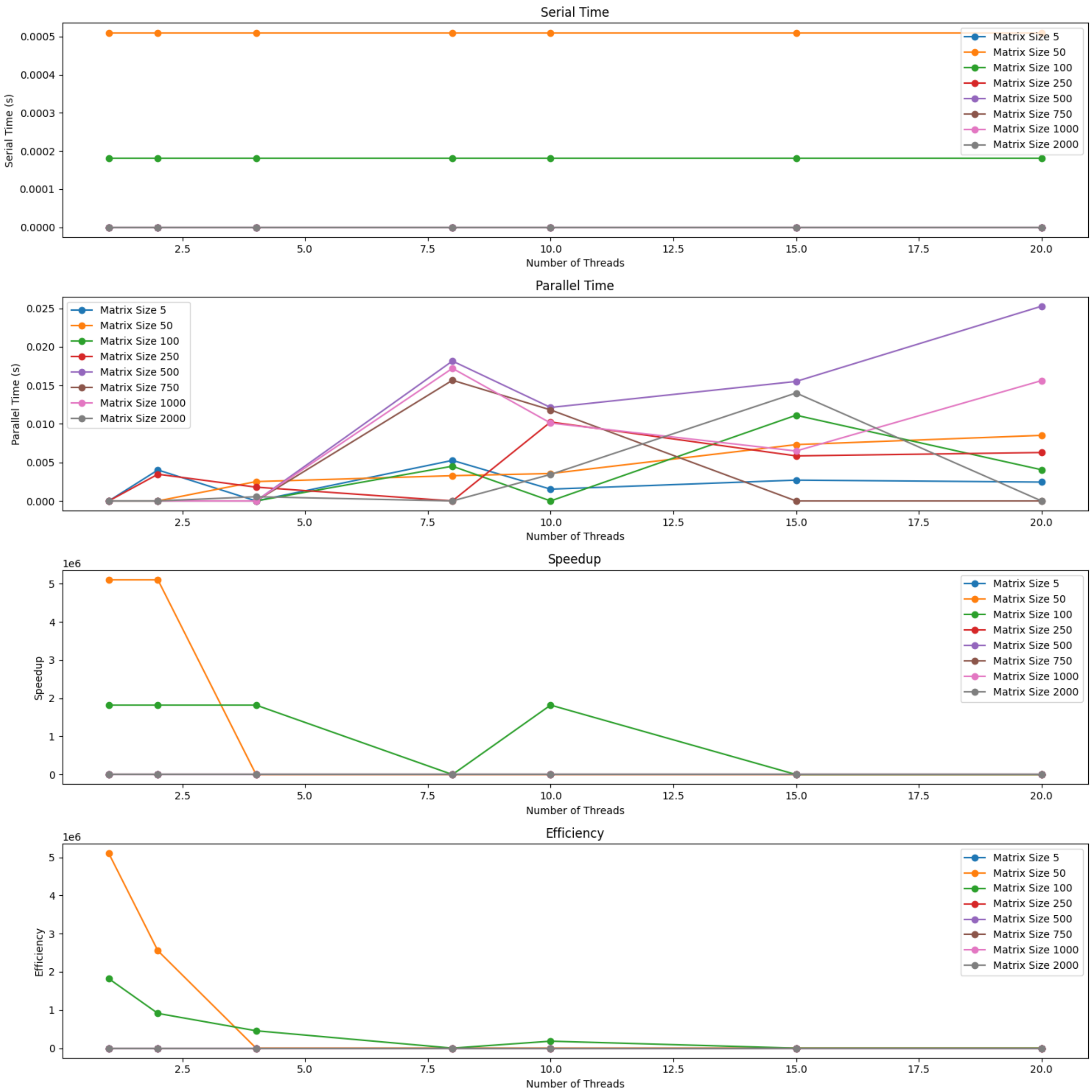



Matrix Size: (100, 100) - Time Comparison
Time (seconds)
Number of Threads
Sequential Time
Parallel Time
Matrix Size: (100, 100) - Speedup
Speedup
Number of Threads
Speedup
Matrix Size: (250, 250) - Time Comparison
Time (seconds)
Number of Threads
Sequential Time
Parallel Time
Matrix Size: (250, 250) - Speedup
Speedup
Number of Threads
Speedup
Matrix Size: (500, 500) - Time Comparison
Time (seconds)
Number of Threads
Sequential Time
Parallel Time
Matrix Size: (500, 500) - Speedup
Speedup
Number of Threads
Speedup



visualization of performance comparison between serial, parallel using NumPY code

In [9]:
```
def plot_matrix_size(df2, matrix_size):
    plt.figure(figsize=(15, 15))

    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=
    plt.title('Serial Time')
    plt.xlabel('Number of Threads')
    plt.ylabel('Serial Time (s)')
    plt.legend()

    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}', marke
    plt.title('Parallel Time')
    plt.xlabel('Number of Threads')
    plt.ylabel('Parallel Time (s)')
    plt.legend()

    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()

    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='
    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)
```



Conclusion

In general, Numpy's matrix multiplication is faster than using loops for matrix operations in Python. NumPy is a powerful numerical computing library that is highly optimized. Which can be observed from the graph. Also we can see that as size of matrix increases parallel processing is faster then squencial and it gets faster as the number of thread increase. So we can say scaling behavior is as expected.

In [ ]:

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.

1. 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.
2. 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.bold text

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

In [2]:
```
def calculate_pi_sequential(terms):
    sum = 0.0
    sign = 1.0

    for i in range(terms):
        term = 1.0 / (2 * i + 1) * sign
        sum += term
        sign = -sign

    return 4.0 * sum

def calculate_pi(start, end):
    sum = 0.0
    sign = 1.0

    for i in range(start, end):
        term = 1.0 / (2 * i + 1) * sign
        sum += term
        sign = -sign

    return sum

def parallel_calculate_pi(terms, threads):
    chunk_size = terms // threads

    with concurrent.futures.ThreadPoolExecutor(max_workers=threads) as executor:
        futures = [executor.submit(calculate_pi, i * chunk_size, (i + 1) * chunk_size) for

    return 4.0 * sum(future.result() for future in concurrent.futures.as_completed(futures)

def main():
    thread_counts = [1, 10, 20]
    term_counts = [100, 1000, 10000, 1000000]

    results = []

    for threads in thread_counts:
        for terms in term_counts:
            start_time = time.time()

            if threads == 1:
                result = calculate_pi_sequential(terms)
            else:
                result = parallel_calculate_pi(terms, threads)

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

            results.append({
                'Threads': threads,
                'Terms': terms,
                'PI': result,
                'Time': elapsed_time
            })

    df = pd.DataFrame(results)
    return df

if __name__ == "__main__":
    df = main()
```

In [3]:
```
df
```

Out[3]:

	Threads	Terms	PI	Time
0	1	100	3.131593	0.000000
1	1	1000	3.140593	0.003346
2	1	10000	3.141493	0.000000
3	1	1000000	3.141592	0.232708
4	10	100	3.131593	0.001878
5	10	1000	3.140593	0.002519
6	10	10000	3.141493	0.003670
7	10	1000000	3.141592	0.315063
8	20	100	4.566072	0.006325
9	20	1000	3.140593	0.004514
10	20	10000	3.141493	0.005029
11	20	1000000	3.141592	0.455092

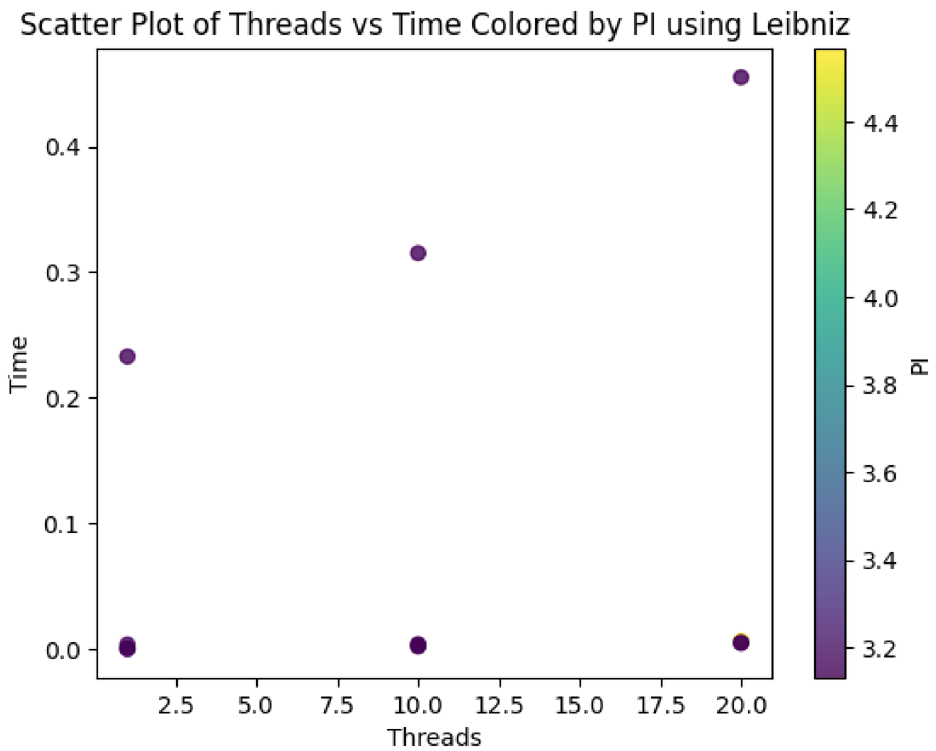
In [4]:
```python
import time
import concurrent.futures
import pandas as pd
import math

def calculate_pi_sequential(terms):
    return math.pi

def calculate_pi(start, end):
    return sum(((-1) ** i) / (2 * i + 1) for i in range(start, end)) * 4.0

def parallel_calculate_pi(terms, threads):
    chunk_size = terms // threads

    with concurrent.futures.ThreadPoolExecutor(max_workers=threads) as executor:
        futures = [executor.submit(calculate_pi, i * chunk_size, (i + 1) * chunk_size) for

    return 4.0 * sum(future.result() for future in concurrent.futures.as_completed(futures)

def main():
    thread_counts = [1, 10, 20]
    term_counts = [100, 1000, 10000, 1000000]

    results = []

    for threads in thread_counts:
        for terms in term_counts:
            start_time = time.time()

            if threads == 1:
                result = calculate_pi_sequential(terms)
            else:
                result = parallel_calculate_pi(terms, threads)

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

            results.append({
                'Threads': threads,
                'Terms': terms,
                'PI': result,
                'Time': elapsed_time
            })

    df = pd.DataFrame(results)
    return df

if __name__ == "__main__":
    df1 = main()
```

In [5]:
```
df1
```

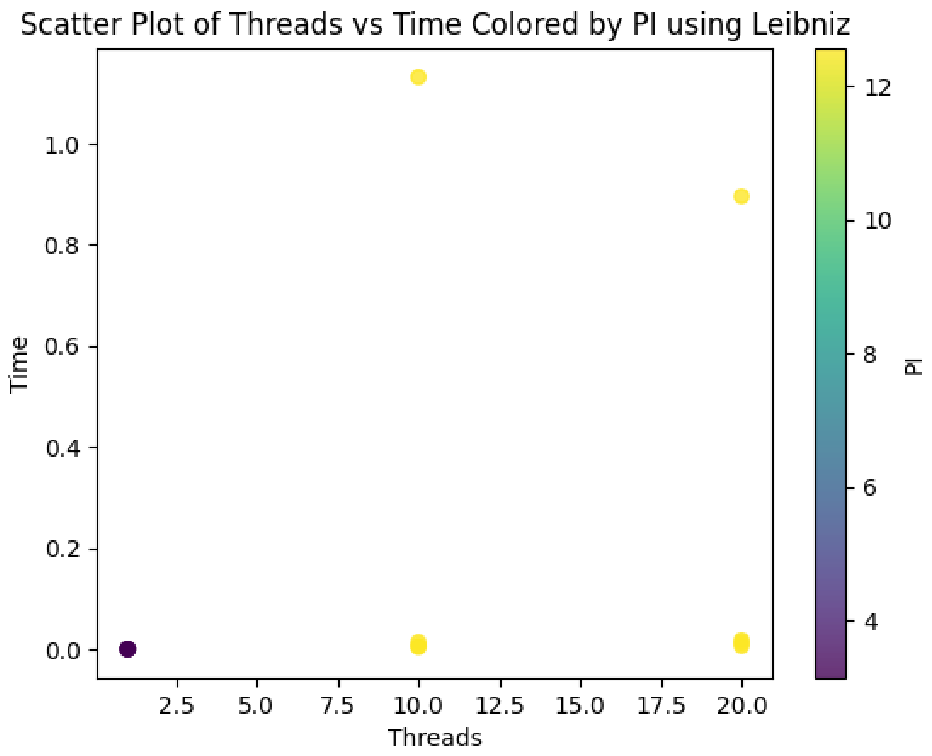
Out[5]:

	Threads	Terms	PI	Time
0	1	100	3.141593	0.000000
1	1	1000	3.141593	0.000000
2	1	10000	3.141593	0.000000
3	1	1000000	3.141593	0.000000
4	10	100	12.526372	0.005517
5	10	1000	12.562371	0.004237
6	10	10000	12.565971	0.013273
7	10	1000000	12.566367	1.131677
8	20	100	12.526372	0.017192
9	20	1000	12.562371	0.006417
10	20	10000	12.565971	0.012102
11	20	1000000	12.566367	0.895726

In [6]:
```
plt.scatter(df['Threads'], df['Time'], c=df['PI'], cmap='viridis', marker='o', alpha=0.8)
plt.xlabel('Threads')
plt.ylabel('Time')
plt.title('Scatter Plot of Threads vs Time Colored by PI using Leibniz')
plt.colorbar(label='PI')
plt.show()
```



In [7]:
```
plt.scatter(df1['Threads'], df1['Time'], c=df1['PI'], cmap='viridis', marker='o', alpha=0.8
plt.xlabel('Threads')
plt.ylabel('Time')
plt.title('Scatter Plot of Threads vs Time Colored by PI using Leibniz')
plt.colorbar(label='PI')
plt.show()
```



In [ ]:

Assignment - 3

The producer-consumer problem is a synchronization challenge in the field of operating systems, particularly in scenarios involving concurrent programming and multi-threading. It revolves around two types of processes.

- Producers: These processes are responsible for generating data or items and placing them in a shared buffer.
- Consumers: These processes retrieve and consume items from the buffer.

The primary goal is to ensure the following conditions are met:

Producers should refrain from producing items if the buffer is full. Consumers should avoid consuming items if the buffer is empty. The central objective is to maintain synchronization between producers and consumers to prevent issues such as data corruption, race conditions, and deadlocks.

- Solution Approach:

A practical solution involves the use of bounded buffers and semaphores, which can be explained as follows:

1. Shared Buffer: A fixed-size buffer is utilized, acting as a common storage space for both producers and consumers.
2. Semaphore for Empty Slots (empty): Initialized to the size of the buffer. Represents the count of empty slots in the buffer. Decreases by producers when they add an item. Decreases by consumers when they remove an item.
3. Semaphore for Full Slots (full): Initialized to 0. Represents the count of filled slots in the buffer. Increased by producers when they add an item. Decreases by consumers when they remove an item.

Example:

Let's consider a scenario in a restaurant where there are chefs (producers) preparing dishes and waiters (consumers) serving these dishes to customers. The shared buffer is the kitchen counter where dishes are temporarily placed before being served.

1. Shared Buffer (Kitchen Counter): Represents the kitchen counter where the prepared dishes are temporarily stored.
2. Semaphore for Empty Serving Plates (empty): Initialized to the maximum capacity of the counter. Indicates the count of empty serving plates on the kitchen counter. Decreases as chefs place prepared dishes on empty plates. Decreases as waiters take dishes from the counter to serve customers.
3. Semaphore for Full Serving Plates (full):Initialized to 0. Represents the count of plates with prepared dishes on the counter. Increases as chefs place dishes on empty plates. Decreases as waiters take dishes from the counter to serve customers. In this analogy, the restaurant ensures that chefs don't prepare more dishes if there are no empty plates, and waiters don't serve if there are no prepared dishes on the counter, effectively managing the flow of food production and service.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import threading
import time
```

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

```
# Declaring Semaphores
mutex = threading.Semaphore()
empty = threading.Semaphore(CAPACITY)
full = threading.Semaphore(0)
```

```
# 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
```

```
# 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
```

```
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
```

Implemantation using if-else and for loop

```
CAPACITY = 10
buffer = [-1 for i in range(CAPACITY)]
```

```
in_index = 0
out_index = 0
```

```
mutex = threading.Semaphore()
empty = threading.Semaphore(CAPACITY)
full = threading.Semaphore(0)
```

```
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)
```

```
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)
```

```
producer = Producer()
consumer = Consumer()
consumer.start()
producer.start()
producer.join()
consumer.join()
```

```
Producer produced: 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
Consumer consumed item: 1
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 : **Restaurant Order Fulfillment System - Producer-Consumer Problem**

- Producers (Chefs): Represented by the Chef thread, they generate items by preparing dishes and place them in a shared buffer (kitchen counter).
- Buffer (Kitchen Counter): Simulates the shared buffer in the producer-consumer problem. It's a queue (kitchen_counter) where Chefs (producers) deposit prepared dishes, and Waiters (consumers) retrieve and serve them.
- Consumers (Waiters): Represented by the Waiter thread, they consume items from the shared buffer (kitchen counter) by taking and

```
import queue
```

```
# Shared resources
kitchen_counter = queue.Queue(maxsize=5)  # Shared buffer (kitchen counter)
empty_plates_semaphore = threading.Semaphore(5)  # Semaphore for empty serving plates
full_plates_semaphore = threading.Semaphore(0)   # Semaphore for full serving plates
```

```
# Producer (Chef) function
class Chef(threading.Thread):
    def run(self):
        for _ in range(10):
            time.sleep(1)  # Simulate time taken to prepare a dish
            empty_plates_semaphore.acquire()  # Decrease count of empty plates
            dish = f'Dish {time.time()}'
            kitchen_counter.put(dish)  # Place the prepared dish on the kitchen counter
            print(f'Chef prepared {dish}')
            full_plates_semaphore.release()  # Increase count of full plates
```

```
# Consumer (Waiter) function
class Waiter(threading.Thread):
    def run(self):
        for _ in range(10):
            full_plates_semaphore.acquire()  # Decrease count of full plates
            dish = kitchen_counter.get()  # Take a dish from the kitchen counter
            print(f'Waiter served {dish}')
            empty_plates_semaphore.release()  # Increase count of empty plates
```

```
# Create threads for chef and waiter
chef_thread = Chef()
waiter_thread = Waiter()

# Start the threads
chef_thread.start()
waiter_thread.start()

# Allow the threads to run for a while (you can interrupt the program manually)
chef_thread.join()
waiter_thread.join()
```

```
Chef prepared Dish 1706694540.4464412
Waiter served Dish 1706694540.4464412
Chef prepared Dish 1706694541.4482815
Waiter served Dish 1706694541.4482815
Chef prepared Dish 1706694542.4495764
Waiter served Dish 1706694542.4495764
Chef prepared Dish 1706694543.4508727
Waiter served Dish 1706694543.4508727
Chef prepared Dish 1706694544.4521263
Waiter served Dish 1706694544.4521263
Chef prepared Dish 1706694545.453453
Waiter served Dish 1706694545.453453
Chef prepared Dish 1706694546.4547088
Waiter served Dish 1706694546.4547088
Chef prepared Dish 1706694547.4556842
Waiter served Dish 1706694547.4556842
Chef prepared Dish 1706694548.4571862
Waiter served Dish 1706694548.4571862
Chef prepared Dish 1706694549.4584432
Waiter served Dish 1706694549.4584432
```

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.

```
import threading
import time
```

```
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
```

```
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")
```

```
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: 17
0, Thread ID 138807072519744 at 2024-02-01 11:55:04: , Time Taken: 0.000328 seconds
1, Thread ID 138807072519744 at 2024-02-01 11:55:04: , Time Taken: 0.000330 seconds
1, Thread ID 138807072519744 at 2024-02-01 11:55:04: , Time Taken: 0.000331 seconds
2, Thread ID 138807072519744 at 2024-02-01 11:55:04: , Time Taken: 0.000332 seconds
3, Thread ID 138807072519744 at 2024-02-01 11:55:04: , Time Taken: 0.000333 seconds
5, Thread ID 138807072519744 at 2024-02-01 11:55:04: , Time Taken: 0.000335 seconds
8, Thread ID 138807072519744 at 2024-02-01 11:55:04: , Time Taken: 0.000335 seconds
13, Thread ID 138807072519744 at 2024-02-01 11:55:04: , Time Taken: 0.000337 seconds
21, Thread ID 138807072519744 at 2024-02-01 11:55:04: , Time Taken: 0.000338 seconds
34, Thread ID 138807072519744 at 2024-02-01 11:55:04: , Time Taken: 0.000340 seconds
55, Thread ID 138807072519744 at 2024-02-01 11:55:04: , Time Taken: 0.000341 seconds
89, Thread ID 138807072519744 at 2024-02-01 11:55:04: , Time Taken: 0.000343 seconds
144, Thread ID 138807072519744 at 2024-02-01 11:55:04: , Time Taken: 0.000344 seconds
233, Thread ID 138807072519744 at 2024-02-01 11:55:04: , Time Taken: 0.000346 seconds
377, Thread ID 138807072519744 at 2024-02-01 11:55:04: , Time Taken: 0.000347 seconds
610, Thread ID 138807072519744 at 2024-02-01 11:55:04: , Time Taken: 0.000348 seconds
987, Thread ID 138807072519744 at 2024-02-01 11:55:04: , Time Taken: 0.000350 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.

In [1]:

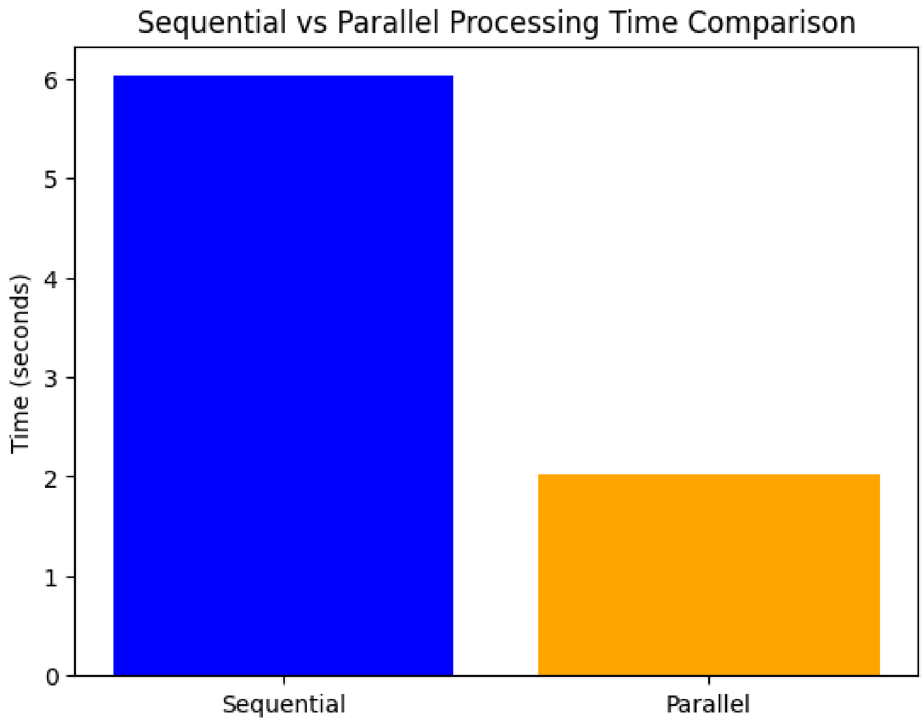
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import threading
import time
import matplotlib.pyplot as plt
```

In [ ]:

```python
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)

# Function to process bill in the clothing section
def process_clothing():
    print("Processing clothing item...")
    time.sleep(0.2)  # Simulating processing time
    return 30  # Cost of each clothing item

# Function to process bill in the grocery section
def process_grocery():
    print("Processing grocery item...")
    time.sleep(0.2)  # Simulating processing time
    return 25  # Cost of each grocery item

# Function to process bill in the utensils section
def process_utensils():
    print("Processing utensils item...")
    time.sleep(0.2)  # Simulating processing time
    return 15  # Cost of each utensils item
```

In [2]:
```python
if __name__ == "__main__":
    # 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")

    # 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)
    ]

    # Start threads
    for thread in threads:
        thread.start()

    # Wait for all threads to finish
    for thread in threads:
        thread.join()

    # Calculate total cost
    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")

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

    plt.bar(labels, times, color=['blue', 'orange'])
    plt.ylabel('Time (seconds)')
    plt.title('Sequential vs Parallel Processing Time Comparison')
    plt.show()
```

```
Processing clothing item...
Processing clothing item...
Processing clothing item...
Processing clothing item...
Processing clothing item...
Processing clothing item...
Processing clothing item...
Processing clothing item...
Processing clothing item...
Processing clothing item...
Processing grocery item...
Processing grocery item...
Processing grocery item...
Processing grocery item...
Processing grocery item...
Processing grocery item...
Processing grocery item...
Processing grocery item...
Processing grocery item...
Processing grocery item...
Processing utensils item...
Processing utensils item...
Processing utensils item...
Processing utensils item...
Processing utensils item...
Processing utensils item...
Processing utensils item...
Processing utensils item...
Processing utensils item...
Processing utensils item...
Total amount to be paid (Sequential): $700.00
Time taken (Sequential): 6.03 seconds

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

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

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

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

Total amount to be paid (Parallel): $700.00
Time taken (Parallel): 2.03 seconds
```



From the graph we can say that sequential processing will take almost thrice the time taken as compared to parallel , since each process is done on different different thread the time taken is less in case of parallel processing.

Assignment - 6

1. Print “Welcome to PDPU from process (processno_totalprocesses)".
2. Apply denoising algorithm to a set of n images with 4 processes. (n=4,8).
3. Analyze time taken by serial and openMPI processes.
4. Try for 100 or more number of images.

In [1]:
```
from mpi4py.futures import MPIPoolExecutor
from mpi4py import MPI
import os
import cv2
import matplotlib.pyplot as plt
import numpy as np
import time
import random
```

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

comm.Barrier()

for i in range(size):
    if rank == i:
        print('Welcome to PDPU From process %d of %d' % (rank, size))
    comm.Barrier()

comm.Barrier()
```

```
Welcome to PDPU From process 0 of 1
```

In [4]:
```
!mpiexec -n 4 python mpi_script.py
```

```
Welcome to PDPU From processes 3 of 4
Welcome to PDPU From processes 1 of 4
Welcome to PDPU From processes 2 of 4
Welcome to PDPU From processes 0 of 4
```

In [5]:
```
!mpiexec -n 8 python mpi_script.py
```

```
Welcome to PDPU From processes 3 of 8
Welcome to PDPU From processes 7 of 8
Welcome to PDPU From processes 5 of 8
Welcome to PDPU From processes 1 of 8
Welcome to PDPU From processes 4 of 8
Welcome to PDPU From processes 2 of 8
Welcome to PDPU From processes 0 of 8
Welcome to PDPU From processes 6 of 8
```

In [9]:
```python
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_im
            cv2.imwrite(os.path.join(denoised_output_folder, f"denoised_image_{i}.jpg"), de
    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 = r"C:\Users\raval\jupyter_notebook\HPC_Lab\crick_data"
    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: 129
Serial Denoising:
Time taken for serial denoising: 65.78 seconds
Parallel Denoising:
Time taken for parallel denoising: 62.18 seconds
```

In [12]:
```
!mpiexec -n 4 python images_100.py
```

```
Number of images: 129
Serial Denoising:
Time taken for serial denoising: 78.91 seconds
Parallel Denoising:
Time taken for parallel denoising: 20.00 seconds
```

In [13]:
```
!mpiexec -n 8 python images_100.py
```

```
Number of images: 129
Serial Denoising:
Time taken for serial denoising: 104.83 seconds
Parallel Denoising:
Time taken for parallel denoising: 13.12 seconds
```

In [15]:
```python
def plot_images_side_by_side(noisy_folder, denoised_folder):
    noisy_image_files = os.listdir(noisy_folder)
    denoised_image_files = os.listdir(denoised_folder)

    noisy_image_files.sort()
    denoised_image_files.sort()

    num_images = min(len(noisy_image_files), len(denoised_image_files), 10)
    fig, axes = plt.subplots(num_images, 2, figsize=(8, num_images * 4))

    for i in range(num_images):
        noisy_image_path = os.path.join(noisy_folder, noisy_image_files[i])
        noisy_image = cv2.imread(noisy_image_path)
        noisy_image = cv2.cvtColor(noisy_image, cv2.COLOR_BGR2RGB)

        denoised_image_path = os.path.join(denoised_folder, denoised_image_files[i])
        denoised_image = cv2.imread(denoised_image_path)
        denoised_image = cv2.cvtColor(denoised_image, cv2.COLOR_BGR2RGB)

        axes[i, 0].imshow(noisy_image)
        axes[i, 0].set_title(f"Noisy Image {i+1}")
        axes[i, 0].axis('off')

        axes[i, 1].imshow(denoised_image)
        axes[i, 1].set_title(f"Denoised Image {i+1}")
        axes[i, 1].axis('off')

    plt.tight_layout()
    plt.show()

noisy_folder = r"C:\Users\raval\jupyter_notebook\HPC_Lab\crick_data\output1\noisy"
denoised_folder = r"C:\Users\raval\jupyter_notebook\HPC_Lab\crick_data\output1\denoised"
plot_images_side_by_side(noisy_folder, denoised_folder)
```


Noisy Image 1
Denoised Image 1
Noisy Image 2
Denoised Image 2
Noisy Image 3
Denoised Image 3
Noisy Image 4
Denoised Image 4

Noisy Image 5

Denoised Image 5

Noisy Image 6

Denoised Image 6

Noisy Image 7

Denoised Image 7


Noisy Image 8

Denoised Image 8

Noisy Image 9

Denoised Image 9

Noisy Image 10

Denoised Image 10

```
In [ ]:
```

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

In [2]:
```
import mpi4py
from mpi4py import MPI
import numpy as np
```

In [ ]:
```
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
```

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

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

In [ ]:
```
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)
```

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

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

In [ ]:
```

```

In [8]:
```
import cv2
```

In [9]:
```python
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()
```

In [13]:
```python
filename = "01.jpg"
image = read_image(filename)
```

In [14]:
```python
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')
```

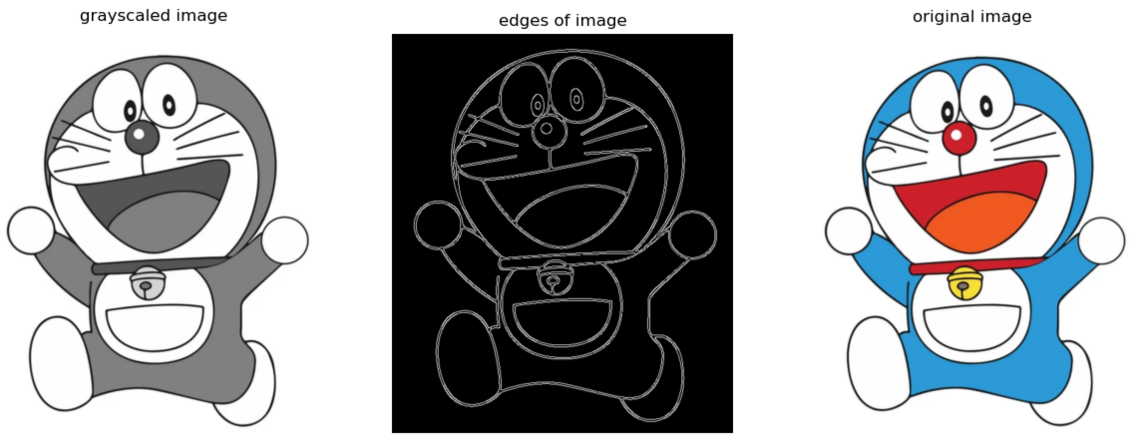
```
rank = 0 , Read the image
```

In [19]:
```python
!mpiexec -n 4 python hpc-7(2).py
```

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

In [17]:
```python
import matplotlib.pyplot as plt
```

In [20]:
```python
# Load the images
image1 = cv2.imread("hpc-7-2-1.png")
image2 = cv2.imread("hpc-7-2-2.png")
image3 = cv2.imread("hpc-7-2-3.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()
```



In [ ]:

In [3]:

In [ ]:

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.

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

In [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
```

In [3]:
```
randNum = np.zeros(1)
```

In [6]:
```
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)
```

In [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
```

In [ ]:

In [ ]:
```
if rank == 0:
    message = "Hello from process 0 (Async)"
    req_send = comm.isend(message, dest=1) # Non-blocking send
    print(f"Process {rank} sent message: {message}")
    time.sleep(1) # Simulate some other task
    req_send.wait() # Wait for the send operation to complete
elif rank == 1:
    req_recv = comm.irecv(source=0) # Non-blocking receive
    time.sleep(0.5) # Simulate some other task
    print(f"Process {rank} waiting to receive message...")
    received_message = req_recv.wait() # Wait for the receive operation to
↪complete
    print(f"Process {rank} received message: {received_message}")
```

In [7]:
```
!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)
```

```
In [ ]:
```

Assignment - 9

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

In [1]:
```
from mpi4py import MPI
import random
import matplotlib.pyplot as plt
import time
```

In [2]:
```
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

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

    terms = 100000000

    start_time = time.time()

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

    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"Process {i} received: Pi = {partial_result}, Time = {partial_time} seconds")

        print("Number of processes:", size)
        print("Estimated Pi:", total_pi)
        print("Execution time:", time.time() - start_time, "seconds")
    else:
        comm.send((partial_pi, time.time() - start_time), dest=0)
```

```
Process 0 calculated: Pi = 3.141592643589326, Time = 45.04786658287048 seconds
Number of processes: 1
Estimated Pi: 3.141592643589326
Execution time: 45.04786658287048 seconds
```

In [3]:
```
!mpiexec -n 2 python Calculate_Pi_value.py
```

```
Process 1 calculated: Pi = -18.813394448175345, Time = 15.578516483306885 seconds
Process 0 calculated: Pi = 21.954987091759833, Time = 16.12114691734314 seconds
Process 1 received: Pi = -18.813394448175345, Time = 15.578516483306885 seconds
Number of processes: 2
Estimated Pi: 3.141592643584488
Execution time: 16.13143539428711 seconds
```

In [4]:
```
!mpiexec -n 4 python Calculate_Pi_value.py
```

```
Process 1 calculated: Pi = -9.894192713487952, Time = 7.745937824249268 seconds
Process 3 calculated: Pi = -8.919201734688878, Time = 8.286001443862915 seconds
Process 2 calculated: Pi = 9.243547576205538, Time = 8.427385330200195 seconds
Process 0 calculated: Pi = 12.711439515567903, Time = 7.98891019821167 seconds
Process 1 received: Pi = -9.894192713487952, Time = 7.745937824249268 seconds
Process 2 received: Pi = 9.243547576205538, Time = 8.427385330200195 seconds
Process 3 received: Pi = -8.919201734688878, Time = 8.286001443862915 seconds
Number of processes: 4
Estimated Pi: 3.1415926435966117
Execution time: 8.424819946289062 seconds
```

In [5]:
```
!mpiexec -n 8 python Calculate_Pi_value.py
```

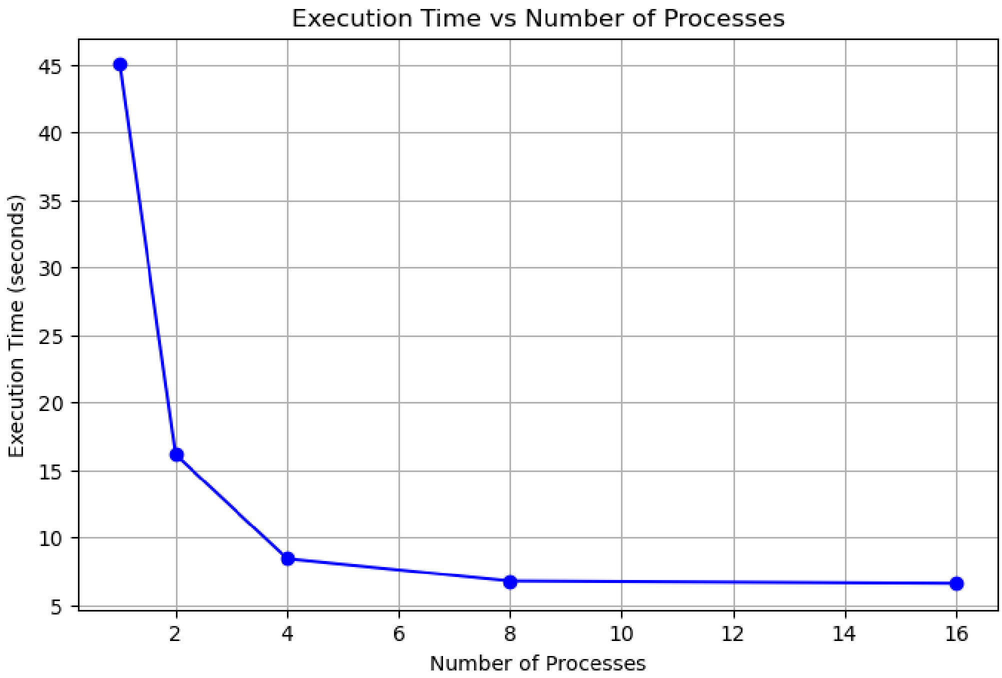
```
Process 7 calculated: Pi = -4.256559977941305, Time = 5.042468786239624 seconds
Process 5 calculated: Pi = -4.399299923327154, Time = 5.090317964553833 seconds
Process 6 calculated: Pi = 4.319461385334904, Time = 6.456605434417725 seconds
Process 1 calculated: Pi = -5.494892790161891, Time = 6.5808985233306885 seconds
Process 4 calculated: Pi = 4.5064163512322155, Time = 6.639330863952637 seconds
Process 3 calculated: Pi = -4.662641756747161, Time = 6.6474997997283936 seconds
Process 2 calculated: Pi = 4.924086190869872, Time = 6.733549118041992 seconds
Process 0 calculated: Pi = 8.205023164331104, Time = 6.762158155441284 seconds
Process 1 received: Pi = -5.494892790161891, Time = 6.5808985233306885 seconds
Process 2 received: Pi = 4.924086190869872, Time = 6.733549118041992 seconds
Process 3 received: Pi = -4.662641756747161, Time = 6.6474997997283936 seconds
Process 4 received: Pi = 4.5064163512322155, Time = 6.639330863952637 seconds
Process 5 received: Pi = -4.399299923327154, Time = 5.090317964553833 seconds
Process 6 received: Pi = 4.319461385334904, Time = 6.456605434417725 seconds
Process 7 received: Pi = -4.256559977941305, Time = 5.042468786239624 seconds
Number of processes: 8
Estimated Pi: 3.1415926435905845
Execution time: 6.762705564498901 seconds
```

In [6]:
```
!mpiexec -n 16 python Calculate_Pi_value.py
```

```
Process 5 calculated: Pi = -2.334733085800057, Time = 2.950575113296509 seconds
Process 14 calculated: Pi = 2.048883162920837, Time = 3.011434316635132 seconds
Process 9 calculated: Pi = -2.1514542620738304, Time = 2.9923923015594482 seconds
Process 1 calculated: Pi = -3.343438528087643, Time = 4.54207181930542 seconds
Process 11 calculated: Pi = -2.102010079259159, Time = 4.207803964614868 seconds
Process 4 calculated: Pi = 2.4242898780367725, Time = 3.8869941234588623 seconds
Process 3 calculated: Pi = -2.5606316774894724, Time = 4.130717992782593 seconds
Process 10 calculated: Pi = 2.1248310598041242, Time = 5.66577935218811 seconds
Process 13 calculated: Pi = -2.064566837527382, Time = 5.360623121261597 seconds
Process 7 calculated: Pi = -2.221819992321798, Time = 5.874985933303833 seconds
Process 8 calculated: Pi = 2.1831425099178574, Time = 5.779357671737671 seconds
Process 15 calculated: Pi = -2.0347399856201327, Time = 5.342623949050903 seconds
Process 6 calculated: Pi = 2.2705782224129933, Time = 5.56093430519104 seconds
Process 2 calculated: Pi = 2.799255131066239, Time = 5.447075843811035 seconds
Process 12 calculated: Pi = 2.0821264731939486, Time = 5.176719427108765 seconds
Process 0 calculated: Pi = 6.021880654414, Time = 3.008913993835449 seconds
Process 1 received: Pi = -3.343438528087643, Time = 4.54207181930542 seconds
Process 2 received: Pi = 2.799255131066239, Time = 5.447075843811035 seconds
Process 3 received: Pi = -2.5606316774894724, Time = 4.130717992782593 seconds
Process 4 received: Pi = 2.4242898780367725, Time = 3.8869941234588623 seconds
Process 5 received: Pi = -2.334733085800057, Time = 2.950575113296509 seconds
Process 6 received: Pi = 2.2705782224129933, Time = 5.56093430519104 seconds
Process 7 received: Pi = -2.221819992321798, Time = 5.874985933303833 seconds
Process 8 received: Pi = 2.1831425099178574, Time = 5.779357671737671 seconds
Process 9 received: Pi = -2.1514542620738304, Time = 2.9923923015594482 seconds
Process 10 received: Pi = 2.1248310598041242, Time = 5.66577935218811 seconds
Process 11 received: Pi = -2.102010079259159, Time = 4.208165645599365 seconds
Process 12 received: Pi = 2.0821264731939486, Time = 5.176719427108765 seconds
Process 13 received: Pi = -2.064566837527382, Time = 5.360623121261597 seconds
Process 14 received: Pi = 2.048883162920837, Time = 3.011434316635132 seconds
Process 15 received: Pi = -2.0347399856201327, Time = 5.342623949050903 seconds
Number of processes: 16
Estimated Pi: 3.141592643587298
Execution time: 6.584681272506714 seconds
```

In [8]:
```
num_processes = [1,2, 4, 8, 16]
execution_times = [45.04786658287048, 16.13143539428711, 8.424819946289062, 6.762705564498901,6.58468127250671
```

In [9]:
```
plt.figure(figsize=(8, 5))
plt.plot(num_processes, execution_times, marker='o', color='blue')
plt.title('Execution Time vs Number of Processes')
plt.xlabel('Number of Processes')
plt.ylabel('Execution Time (seconds)')
plt.grid(True)
plt.show()
```



In [ ]:

Assignment - 10

1. 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
   - Bcast function
   - Scatter function
   - Gather function

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

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

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: 407
```

In [3]:
```
!mpiexec -n 4 python Bcast.py
```

```
Root process (Rank 0) is broadcasting data to other processes...
Process 0 received data: 20
Process 1 is waiting to receive broadcasted data from the root process (Rank 0)
Process 1 received data: 20
Process 2 is waiting to receive broadcasted data from the root process (Rank 0)
Process 2 received data: 20
Process 3 is waiting to receive broadcasted data from the root process (Rank 0)
Process 3 received data: 20
```

In [4]:
```
comm = MPI.COMM_WORLD
rank = comm.Get_rank()
size = comm.Get_size()

if rank == 0:
    print("Root process (Rank 0) is scattering data to other processes...")
else:
    print("Process", rank, "is waiting to receive scattered data from the root process (Ran

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])
```

```
Root process (Rank 0) is scattering data to other processes...
Process 0 received data: 0
```

In [5]:
```
!mpiexec -n 4 python Scatter.py
```

```
Root process (Rank 0) is scattering data to other processes...
Process 0 received data: 0
Process 1 is waiting to receive scattered data from the root process (Rank 0)
Process 1 received data: 10
Process 2 is waiting to receive scattered data from the root process (Rank 0)
Process 2 received data: 20
Process 3 is waiting to receive scattered data from the root process (Rank 0)
Process 3 received data: 30
```

In [6]:
```
comm = MPI.COMM_WORLD
rank = comm.Get_rank()
size = comm.Get_size()

local_sum = np.random.randint(0, 100)

if rank == 0:
    print("Root process (Rank 0) is gathering local sums from other processes...")

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("Root process (Rank 0) gathered the following local sums:", global_sums)
```

```
Root process (Rank 0) is gathering local sums from other processes...
Root process (Rank 0) gathered the following local sums: [88]
```

In [7]:
```
!mpiexec -n 4 python Gather.py
```

```
Root process (Rank 0) is gathering local sums from other processes...
Root process (Rank 0) gathered the following local sums: [79 32 30 23]
```

In [ ]:
```

```

Assignment - 11

1. Describe Canon's Matrix Multiplication algorithm.
2. Implement Canon's Matrix Multiplication using collective communication.
3. Analyze the efficiency of the code.

- Canon's Matrix Multiplication is an algorithm for multiplying matrices it helps with to distribute the computation across multiple processors in a parallel or distributed computing environment it is useful when dealing with very large matrices that cannot be efficiently handled by a single processor.
- Canon's algorithm works as follows

1. Divide each matrix into submatrices.
2. Distribute these submatrices across the processors in a grid-like fashion.
3. Each processor computes the partial products of its assigned submatrices iteratively shift the submatrices horizontally and vertically, so that each processor multiplies its submatrices with the corresponding ones from neighboring processors.
4. Repeat the shifting and multiplication steps until each processor has performed all necessary multiplications.
5. Accumulate the partial results to obtain the final product matrix.

In [1]:
```
from mpi4py import MPI
import numpy as np
import time
import matplotlib.pyplot as plt
```

In [2]:

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

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")
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()

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_multiplicatio
```

```
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:
[[38924 38988 40084 ... 39540 39911 40176]
 [39181 39193 39492 ... 39247 39309 40381]
 [39406 39285 39590 ... 39814 39647 39856]
 ...
 [39500 40874 40270 ... 40317 40574 40609]
 [40338 41030 41022 ... 40296 41134 41550]
 [38284 38920 39233 ... 38895 38968 38838]]
Broadcasting time: 0.03937983512878418 seconds
Matrix multiplication time: 42.12184238433838 seconds
```

In [3]:
```
!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:
[[42017 41045 42732 ... 41943 40837 42255]
 [40844 40173 40798 ... 41069 39922 39766]
 [40892 39590 40565 ... 40460 39376 39704]
 ...
 [40474 40181 40543 ... 40339 39739 40542]
 [41234 40700 41083 ... 40461 40372 39411]
 [41092 40366 41294 ... 40061 39890 39640]]
Broadcasting time: 0.060237884521484375 seconds
Matrix multiplication time: 11.654186248779297 seconds
```

In [5]:
```
!mpiexec -n 8 python MPI_Scatter_Gather.py
```

```
Rank 7: Starting execution
Rank 7: Broadcasting matrices A and B
Rank 7: Matrices A and B broadcasted
Rank 3: Starting execution
Rank 3: Broadcasting matrices A and B
Rank 3: 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 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:
[[40325 40191 41154 ... 40507 40363 39973]
 [39863 39909 41875 ... 41396 39875 40330]
 [40414 39831 40746 ... 40748 40352 39065]
 ...
 [39360 40292 41247 ... 40491 40832 40227]
 [41656 41171 43231 ... 42439 41473 41172]
 [39526 38647 40187 ... 39084 38608 38932]]
Broadcasting time: 0.10047459602355957 seconds
Matrix multiplication time: 10.563331127166748 seconds
Rank 4: Starting execution
Rank 4: Broadcasting matrices A and B
Rank 4: Matrices A and B broadcasted
```

In [4]:
```python
from mpi4py import MPI
import numpy as np
import time

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

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")
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_multiplicatio
```

```
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:
[[40916 40038 41686 ... 40639 42268 40762]
 [39408 38656 40396 ... 39191 40604 39282]
 [40379 39155 41555 ... 40121 41890 39871]
 ...
 [41187 40173 41164 ... 41221 41924 40741]
 [41633 40312 42298 ... 41181 42086 40899]
 [39550 38358 40014 ... 40075 40537 39391]]
Broadcasting time: 0.016689777374267578 seconds
Gathering time: 0.002426624298095703 seconds
Matrix multiplication time: 42.118860960006714 seconds
```

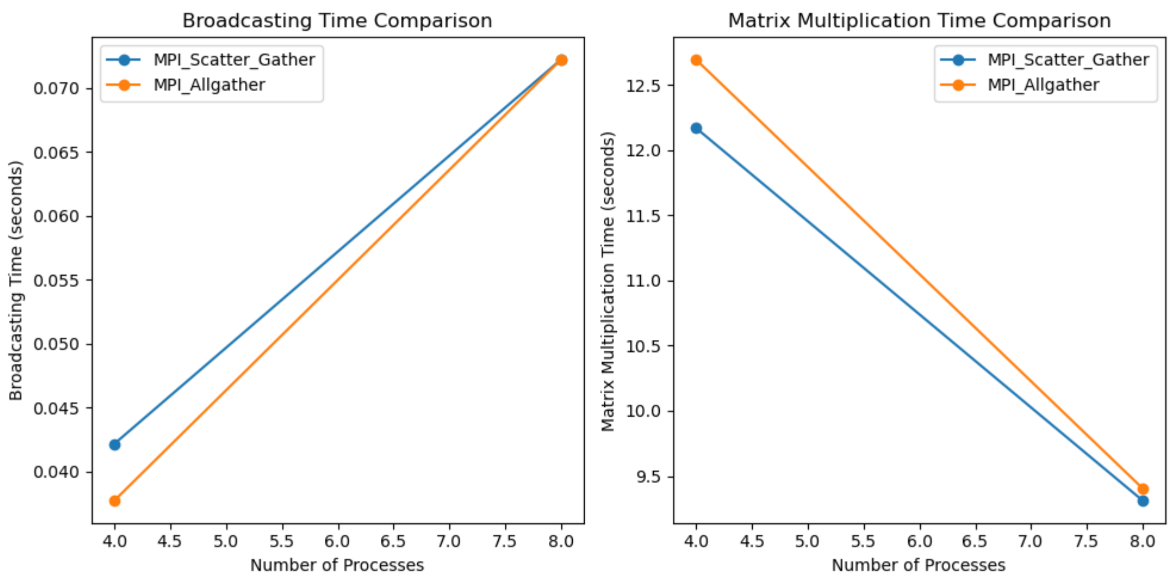
In [6]:
```
!mpiexec -n 4 python MPI_Allgather.py
```

```
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 3: Starting execution
Rank 3: Broadcasting matrices A and B
Rank 3: 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:
[[41981 42298 42010 ... 41927 40632 40686]
 [40844 42040 41133 ... 41613 40279 41009]
 [40145 40810 39721 ... 40101 39097 39790]
 ...
 [40915 40933 40913 ... 40927 39648 40175]
 [40807 41294 40486 ... 41782 40666 40397]
 [40110 40841 40445 ... 41008 40574 40400]]
Broadcasting time: 0.03273797035217285 seconds
Gathering time: 0.01018381118774414 seconds
Matrix multiplication time: 11.341147899627686 seconds
```

In [7]:
```
!mpiexec -n 8 python MPI_Allgather.py
```

```
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 7: Starting execution
Rank 7: Broadcasting matrices A and B
Rank 7: 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:
[[38826 38685 39926 ... 40178 39925 38546]
 [41287 40079 40148 ... 41615 41193 41219]
 [39324 39803 40427 ... 40839 40679 40418]
 ...
 [39700 39258 40242 ... 40461 39879 39241]
 [39591 39742 40000 ... 40148 39897 40145]
 [40836 40086 40435 ... 41049 40958 40317]]
Broadcasting time: 0.09126925468444824 seconds
Gathering time: 0.2480170726776123 seconds
Matrix multiplication time: 10.671711921691895 seconds
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 3: Starting execution
Rank 3: Broadcasting matrices A and B
Rank 3: Matrices A and B broadcasted
```

In [8]:
```
scatter_gather_processes = [4, 8]
scatter_gather_broadcasting_time = [0.04213356971740723, 0.07224988937377793]
scatter_gather_multiplication_time = [12.16959547996521, 9.310021162033081]

allgather_processes = [4, 8]
allgather_broadcasting_time = [0.037689924240112305, 0.0722506046295166 ]
allgather_multiplication_time = [12.694447040557861, 9.402881145477295]

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

plt.subplot(1, 2, 1)
plt.plot(scatter_gather_processes, scatter_gather_broadcasting_time, marker='o', label='MPI
plt.plot(allgather_processes, allgather_broadcasting_time, marker='o', label='MPI_Allgather
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='M
plt.plot(allgather_processes, allgather_multiplication_time, marker='o', label='MPI_Allgath
plt.xlabel('Number of Processes')
plt.ylabel('Matrix Multiplication Time (seconds)')
plt.title('Matrix Multiplication Time Comparison')
plt.legend()

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



1. MPI_Scatter_Gather:

- With 4 processes, the broadcasting time is lower compared to MPI_Allgather, but the matrix multiplication time is higher.
- With 8 processes, the broadcasting time increases slightly, but the matrix multiplication time decreases significantly compared to 4 processes.

2. MPI_Allgather:

- With 4 processes, the broadcasting time is slightly higher compared to MPI_Scatter_Gather, but the gathering time is introduced. However, the matrix multiplication time is comparable to MPI_Scatter_Gather.
- With 8 processes, the broadcasting time is slightly higher compared to MPI_Scatter_Gather, and the gathering time becomes noticeable. However, the matrix multiplication time is the lowest among all configurations, indicating better parallel efficiency.

3. Conclusion:

- MPI_Scatter_Gather might be more efficient for smaller-scale parallelization, where the gathering is not significant compared to the reduction in matrix multiplication time.

- MPI_Allgather becomes more efficient as the number of processes increases, especially for larger-scale parallelization, due to its better load balancing and reduced communication-to-computation ratio.

Assignment - 12

Q1) Write about derived data types used in MPI programming.

Derived data types in MPI (Message Passing Interface) programming are user-defined data structures that allow for more complex data communication than simple built-in types like integers or floats. These types are particularly useful when dealing with structured data such as arrays, structs, or nested data types.

Q2) Steps to create and use derived data types

Here are the steps to create and use derived data types in MPI programming: (1) Define the structure: First, define the structure of your derived data type. This could be a simple struct or a more complex nested structure. (2)

Create the MPI datatype: Use MPI functions to create a new MPI datatype that corresponds to your defined structure. You can specify the arrangement of data within your structure and how it should be communicate(2) d.

Use the MPI datatype: Once you have created the derived datatype, you can use it in MPI communication functions to send and receive data.

Q3) Write its uses.

Derived data types are useful in MPI programming for several reasons: (1) Efficiency: By defining the layout of your data explicitly, you can optimize communication by minimizing the amount of data that needs to be transferred.(2)

Abstraction: Derived data types allow you to abstract away the details of the underlying data structure, making your code cleaner and more modula(3) r.

Complex data structures: MPI's built-in communication functions are designed for simple data types like integers and floats. Derived data types allow you to communicate more complex data structures efficiently.

Q4) Implement communication of derived data using one suitable example.

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

# Create MPI communicator
comm = MPI.COMM_WORLD
rank = comm.Get_rank()
size = comm.Get_size()

# Define the structure of the data
class Particle:
    def __init__(self, x, y):
        self.x = x
        self.y = y

# Data to be sent from rank 0
send_data = None
if rank == 0:
    send_data = Particle(64, 108)

# Scatter data from rank 0 to other processes
recv_data = comm.scatter([send_data] * size, root=0)

# Print received data on each process
print(f"Process {rank} received data: x={recv_data.x}, y={recv_data.y}")

MPI.Finalize()
```

```
Process 0 received data: x=64, y=108
```

In [3]:
```
!mpiexec -n 4 python 12.py
```

```
Process 0 received data: x=64, y=108
Process 3 received data: x=64, y=108
Process 2 received data: x=64, y=108
Process 1 received data: x=64, y=108
```

In [ ]:
```
```

HPC ASSIGNMENT 13

lshw:

```
user@LAPTOP-6QVE9SBJ:~$ sudo lshw
laptop-6qve9sbj
    description: Computer
    width: 64 bits
    capabilities: smp vsyscall32
  *-core
       description: Motherboard
       physical id: 0
     *-memory
          description: System memory
          physical id: 1
          size: 4GiB
     *-cpu
          product: Intel(R) Core(TM) i5-8265U CPU @ 1.60GHz
          vendor: Intel Corp.
          physical id: 2
          bus info: cpu@0
          version: 6.142.12
          width: 64 bits
          capabilities: fpu fpu_exception wp 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 x86-64 constant_tsc arch_perfmon rep_good nopl xtopology cpuid pni pc
lmulqdq vmx ssse3 fma cx16 pdcm pcid sse4_1 sse4_2 movbe popcnt aes xsave avx f16c rdrand hypervisor lahf_lm abm 3dnowpr
efetch invpcid_single ssbd ibrs ibpb stibp ibrs_enhanced tpr_shadow vnmi ept vpid ept_ad fsgsbase bmi1 avx2 smep bmi2 er
ms invpcid rdseed adx smap clflushopt xsaveopt xsavec xgetbv1 xsaves md_clear flush_l1d arch_capabilities
          configuration: microcode=4294967295
     *-generic
          description: System peripheral
```

user@LAPTOP-6QVE9SBJ:~$ sudo lshw

[sudo] password for user:

laptop-6qve9sbj

description: Computer

width: 64 bits

capabilities: smp vsyscall32

*-core

description: Motherboard

physical id: 0

*-memory

description: System memory

physical id: 1

size: 4GiB

*-cpu

product: Intel(R) Core(TM) i5-8265U CPU @ 1.60GHz

vendor: Intel Corp.

physical id: 2

bus info: cpu@0

version: 6.142.12

width: 64 bits

capabilities: fpu fpu_exception wp 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 x86-64 constant_tsc arch_perfmon rep_good nopl xtopology cpuid pni pclmulqdq vmx ssse3 fma cx16 pdcm pcid sse4_1 sse4_2 movbe popcnt aes xsave avx f16c rdrand hypervisor lahf_lm abm 3dnowprefetch invpcid_single ssbd ibrs ibpb stibp ibrs_enhanced tpr_shadow vnmi ept vpid ept_ad fsgsbase bmi1 avx2 smep bmi2 erms invpcid rdseed adx smap clflushopt xsaveopt xsavec xgetbv1 xsaves flush_l1d arch_capabilities

configuration: microcode=4294967295

*-scsi:0

description: SCSI storage controller

product: Virtio console

vendor: Red Hat, Inc.

physical id: 3

bus info: pci@e91b:00:00.0

version: 01

width: 64 bits

clock: 33MHz

capabilities: scsi msix 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

description: 3D controller

product: Microsoft Corporation

vendor: Microsoft Corporation

physical id: 4

bus info: pci@efe4:00:00.0

version: 00

width: 32 bits

clock: 33MHz

capabilities: bus_master cap_list

configuration: driver=dxgkrnl latency=0

resources: irq:0

*-generic

description: System peripheral

product: Virtio file system

vendor: Red Hat, Inc.

physical id: 0

bus info: pci@f482:00:00.0

version: 01

width: 64 bits

clock: 33MHz

capabilities: msix 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-dffffffff

*-virtio1 UNCLAIMED

description: Virtual I/O device

physical id: 0

bus info: virtio@1

configuration: driver=virtiofs

*-pnp00:00

product: PnP device PNP0b00

physical id: 5

capabilities: pnp

configuration: driver=rtc_cmos

*-scsi:1

physical id: 6

logical name: scsi0

*-disk:0

description: SCSI Disk

product: Virtual Disk

vendor: Linux

physical id: 0.0.0

bus info: scsi@0:0.0.0

logical name: /dev/sda

version: 1.0

size: 388MiB

capabilities: extended_attributes large_files huge_files extents ext2 initialized

configuration: ansiversion=5 filesystem=ext2 logicalsectorsize=512 sectorsize=512 state=clean

*-disk:1

description: Linux swap volume

product: Virtual Disk

vendor: Msft

physical id: 0.0.1

bus info: scsi@0:0.0.1

logical name: /dev/sdb

version: 1

serial: 6fe43f8b-4429-46c1-8444-944420a5d6a9

size: 1GiB

capacity: 1GiB

capabilities: swap initialized

configuration: ansiversion=5 filesystem=swap logicalsectorsize=512 pagesize=4096 sectorsize=4096

*-disk:2

description: EXT4 volume

product: Virtual Disk

vendor: Linux

physical id: 0.0.2

bus info: scsi@0:0.0.2

logical name: /dev/sdc

logical name: /

logical name: /mnt/wslg/distro

logical name: /snap

version: 1.0

serial: 54b61250-d471-4149-bd74-bde99aac5bd8

size: 1TiB

capabilities: journaled extended_attributes large_files huge_files dir_nlink recover 64bit extents ext4 ext2 initialized

configuration: ansiversion=5 created=2024-03-13 10:42:07 filesystem=ext4 lastmountpoint=/distro logicalsectorsize=512 modified=2024-03-13 10:42:37 mount.fstype=ext4 mount.options=rw,relatime,discard,errors=remount-ro,data=ordered mounted=2024-03-13 10:42:37 sectorsize=4096 state=mounted

*-usbhost:0

product: USB/IP Virtual Host Controller

vendor: Linux 5.15.146.1-microsoft-standard-WSL2 vhci_hcd

physical id: 1

bus info: usb@1

logical name: usb1

version: 5.15

capabilities: usb-2.00

configuration: driver=hub slots=8 speed=480Mbit/s

*-usbhost:1

product: USB/IP Virtual Host Controller

vendor: Linux 5.15.146.1-microsoft-standard-WSL2 vhci_hcd

physical id: 2

bus info: usb@2

logical name: usb2

version: 5.15

capabilities: usb-3.00

configuration: driver=hub slots=8 speed=5000Mbit/s

*-network

description: Ethernet interface

physical id: 3

logical name: eth0

serial: 00:15:5d:97:fb:b6

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=172.17.39.183 link=yes multicast=yes speed=10Gbit/s

lsusb:

List connected USB devices.

Input: -lsusb

Output: -

```
user@LAPTOP-6QVE9SBJ:~$ 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 installed PCI devices.

Input: - lspci

Output: -

```
user@LAPTOP-6QVE9SBJ:~$ lspci
e91b:00:00.0 SCSI storage controller: Red Hat, Inc. Virtio console (rev 01)
efe4:00:00.0 3D controller: Microsoft Corporation Device 008e
f482:00:00.0 System peripheral: Red Hat, Inc. Virtio file system (rev 01)
```

lsblk:

List block devices and their attributes.

Input: - lsblk

Output: -

```
user@LAPTOP-6IN5E1D3:~$ 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
                               /
user@LAPTOP-6IN5E1D3:~$
```

lscpu:

```
user@LAPTOP-6QVE9SBJ:~$ 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):                  8
  On-line CPU(s) list:   0-7
Vendor ID:               GenuineIntel
  Model name:            Intel(R) Core(TM) i5-8265U CPU @ 1.60GHz
    CPU family:          6
    Model:               142
    Thread(s) per core:  2
    Core(s) per socket:  4
    Socket(s):           1
    Stepping:            12
    BogoMIPS:            3600.01
    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 arch_perfmon rep_good nopl xtopology cpuid pni pclmulqdq vmx s                         sse3 fma cx16 pdcm pcid sse4_1 s
se4_2 movbe popcnt aes xsave avx f16c rdrand hypervisor lahf_lm abm 3dno                         wprefetch invpcid_single ssbd ibrs ibpb stibp ibrs_enhanced tpr_shadow vnmi ept vpid ept_
ad fsgsbase bmi                         1 avx2 smep bmi2 erms invpcid rdseed adx smap clflushopt xsaveopt xsavec xgetbv1 xsaves md_clear flush_l                         1d arch_capabilit
ies
Virtualization features:
  Virtualization:        VT-x
  Hypervisor vendor:     Microsoft
  Virtualization type:   full
Caches (sum of all):
  L1d:                   128 KiB (4 instances)
  L1i:                   128 KiB (4 instances)
  L2:                    1 MiB (4 instances)
  L3:                    6 MiB (1 instance)
Vulnerabilities:
  Gather data sampling:  Unknown: Dependent on hypervisor status
  Itlb multihit:         KVM: Mitigation: VMX disabled
  L1tf:                  Not affected
  Mds:                   Not affected
  Meltdown:              Not affected
  Mmio stale data:       Vulnerable: Clear CPU buffers attempted, no microcode; SMT Host state unknown
  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/swapgs barriers and __user pointer sanitization
  Spectre v2:            Mitigation; Enhanced IBRS, IBPB conditional, RSB filling, PBRSB-eIBRS SW sequence
  Srbds:                 Unknown: Dependent on hypervisor status
  Tsx async abort:       Not affected
```

df:

Show disk space usage and availability.

Input: -df

Output: -

```
user@LAPTOP-6QVE9SBJ:~$ df
Filesystem      1K-blocks      Used  Available Use% Mounted on
none              1968164         4    1968160   1% /mnt/wsl
none            248896836 202486432   46410404  82% /usr/lib/wsl/drivers
none              1968164         0    1968164   0% /usr/lib/modules
none              1968164         0    1968164   0% /usr/lib/modules/5.15.146.1-microsoft-standard-WSL2
/dev/sdc       1055762868   1081364 1000978032   1% /
none              1968164        80    1968084   1% /mnt/wslg
none              1968164         0    1968164   0% /usr/lib/wsl/lib
rootfs            1964904      1884    1963020   1% /init
none              1968164       768    1967396   1% /run
none              1968164         0    1968164   0% /run/lock
none              1968164         0    1968164   0% /run/shm
tmpfs                4096         0       4096   0% /sys/fs/cgroup
none              1968164        76    1968088   1% /mnt/wslg/versions.txt
none              1968164        76    1968088   1% /mnt/wslg/doc
A:\             243268604 175095092   68173512  72% /mnt/a
C:\             248896836 202486432   46410404  82% /mnt/c
D:\             489300988 401430308   87870680  83% /mnt/d
G:\              15728640   4597088   11131552  30% /mnt/g
V:\             244189180 197341692   46847488  81% /mnt/v
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
```

dmidecode:

Decode and display DMI table information.

Input: - dmidecode

Output: -

```
user@LAPTOP-6QVE9SBJ:~$ dmidecode
# dmidecode 3.3
Scanning /dev/mem for entry point.
/dev/mem: Permission denied
```

ip a:

Show IP addresses and network information for all interfaces.

Input: - ip a

Output: -

```
user@LAPTOP-6QVE9SBJ:~$ 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 1400 qdisc mq state UP group default qlen 1000
    link/ether 00:15:5d:97:fb:b6 brd ff:ff:ff:ff:ff:ff
    inet 172.17.39.183/20 brd 172.17.47.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::215:5dff:fe97:fbb6/64 scope link
       valid_lft forever preferred_lft forever
```

top:

Display real-time system resource usage.

Input: -top

Output: -

```
user@LAPTOP-6QVE9SBJ:~$ top
top - 10:59:14 up 17 min,  1 user,  load average: 0.00, 0.01, 0.00
Tasks:  31 total,   1 running,  30 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
MiB Mem :   3844.1 total,   2992.9 free,    501.2 used,    350.0 buff/cache
MiB Swap:   1024.0 total,   1024.0 free,      0.0 used.   3129.5 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
    497 root      20   0   44212  37864  10060 S   0.3   1.0   0:03.33 python3
```

htop:

Interactive version of top with enhanced visualization.

Input: -htop

Output: -

```
user@LAPTOP-6QVE9SBJ: ~

    0[                                  0.0%]   4[|                                 0.7%]
    1[                                  0.0%]   5[                                  0.0%]
    2[                                  0.0%]   6[                                  0.0%]
    3[                                  0.0%]   7[                                  0.0%]
  Mem[|||||||||||||               436M/3.75G]   Tasks: 31, 20 thr; 1 running
  Swp[                              0K/1.00G]   Load average: 0.19 0.07 0.02
                                                Uptime: 00:00:53

    PID USER       PRI  NI  VIRT   RES   SHR S CPU%▽MEM%   TIME+  Command
      1 root        20   0  161M 11108  8200 S  0.0  0.3  0:00.46 /sbin/init
      2 root        20   0  2280  1300  1188 S  0.0  0.0  0:00.00 /init
     10 root        20   0  2280     4     0 S  0.0  0.0  0:00.00 plan9 --control-socket 6 --log-level 4 --server-fd 7 --pipe-fd 9 --l
     11 root        20   0  2280     4     0 S  0.0  0.0  0:00.00 plan9 --control-socket 6 --log-level 4 --server-fd 7 --pipe-fd 9 --l
     12 root        20   0  2280  1300  1188 S  0.0  0.0  0:00.00 /init
     42 root        19  -1 39472 14456 13488 S  0.0  0.4  0:00.09 /lib/systemd/systemd-journald
     62 root        20   0 22088  5868  4420 S  0.0  0.1  0:00.13 /lib/systemd/systemd-udevd
     75 root        20   0  4492   148     0 S  0.0  0.0  0:00.00 snapfuse /var/lib/snapd/snaps/bare_5.snap /snap/bare/5 -o ro,nodev,a
     77 root        20   0  4624   196    44 S  0.0  0.0  0:00.00 snapfuse /var/lib/snapd/snaps/gtk-common-themes_1535.snap /snap/gtk-
     78 root        20   0  4760  1836  1296 S  0.0  0.0  0:01.10 snapfuse /var/lib/snapd/snaps/core22_864.snap /snap/core22/864 -o ro
     80 root        20   0  5024  1776  1228 S  0.0  0.0  0:03.97 snapfuse /var/lib/snapd/snaps/snapd_20290.snap /snap/snapd/20290 -o
     82 root        20   0  4956  1720  1228 S  0.0  0.0  0:01.46 snapfuse /var/lib/snapd/snaps/ubuntu-desktop-installer_1276.snap /sn
     90 systemd-r   20   0 25532 12620  8424 S  0.0  0.3  0:00.19 /lib/systemd/systemd-resolved
    115 root        20   0  4304  2700  2460 S  0.0  0.1  0:00.00 /usr/sbin/cron -f -P
    116 messagebu   20   0  8592  4668  4136 S  0.0  0.1  0:00.04 @dbus-daemon --system --address=systemd: --nofork --nopidfile --syst
    122 root        20   0 30096 19196 10416 S  0.0  0.5  0:00.12 /usr/bin/python3 /usr/bin/networkd-dispatcher --run-startup-triggers
    133 syslog      20   0  217M  5296  4480 S  0.0  0.1  0:00.01 /usr/sbin/rsyslogd -n -iNONE
    135 root        20   0 1649M 34980 18012 S  0.0  0.9  0:00.47 /usr/lib/snapd/snapd
    140 root        20   0 15324  7396  6452 S  0.0  0.2  0:00.12 /lib/systemd/systemd-logind
    151 syslog      20   0  217M  5296  4480 S  0.0  0.1  0:00.00 /usr/sbin/rsyslogd -n -iNONE
    152 syslog      20   0  217M  5296  4480 S  0.0  0.1  0:00.00 /usr/sbin/rsyslogd -n -iNONE
F1Help  F2Setup F3SearchF4FilterF5Tree  F6SortByF7Nice -F8Nice +F9Kill  F10Quit
```

nvidia-smi:

Provide management for NVIDIA GPU devices.

Input: - nvidia-smi

Output: -

```
user@LAPTOP-6QVE9SBJ:~$ nvidia-smi
Command 'nvidia-smi' not found, but can be installed with:
sudo apt install nvidia-utils-390         # version 390.157-0ubuntu0.22.04.2, or
sudo apt install nvidia-utils-418-server  # version 418.226.00-0ubuntu5~0.22.04.1
sudo apt install nvidia-utils-450-server  # version 450.248.02-0ubuntu0.22.04.1
sudo apt install nvidia-utils-470         # version 470.223.02-0ubuntu0.22.04.1
sudo apt install nvidia-utils-470-server  # version 470.223.02-0ubuntu0.22.04.1
sudo apt install nvidia-utils-525         # version 525.147.05-0ubuntu0.22.04.1
sudo apt install nvidia-utils-525-server  # version 525.147.05-0ubuntu0.22.04.1
sudo apt install nvidia-utils-535         # version 535.129.03-0ubuntu0.22.04.1
sudo apt install nvidia-utils-535-server  # version 535.129.03-0ubuntu0.22.04.1
sudo apt install nvidia-utils-510         # version 510.60.02-0ubuntu1
sudo apt install nvidia-utils-510-server  # version 510.47.03-0ubuntu3
```

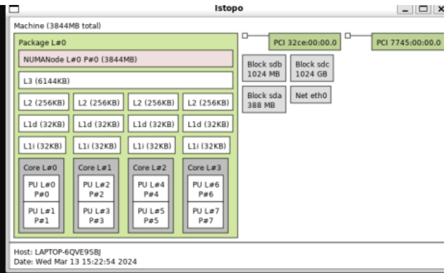
lstopo:

Generate graphical system topology representation.

Input: -lstopo

Output: -

```
user@LAPTOP-6QVE9SBJ:~$ lstopo

Keyboard shortcuts:
 Zooming, scrolling and closing:
  Zoom-in or out ...................... + -
  Reset scale to default .............. 1
  Try to fit scale to window .......... F
  Resize window to the drawing ........ r
  Toggle auto-resizing of the window .. R
  Scroll vertically ................... Up Down PageUp PageDown
  Scroll horizontally ................. Left Right Ctrl+PageUp/Down
  Scroll to the top-left corner ....... Home
  Scroll to the bottom-right corner ... End
  Refresh the topology ................ F5
  Show this help ...................... h H ?
  Exit ................................ q Q Esc
 Configuration tweaks:
  Toggle factorizing or collapsing .... f
  Switch display mode for indexes ..... i
  Toggle displaying of object text .... t
  Toggle displaying of obj attributes . a
  Toggle displaying of CPU kinds ...... k
  Toggle color for disallowed objects . d
  Toggle color for binding objects .... b
  Toggle displaying of legend lines ... l
  Export to file with current config .. E
```



perf:

Collect and analyze performance data.

Input: -perf

Output: -

```
user@LAPTOP-6QVE9SBJ:~$ perf
WARNING: perf not found for kernel 5.15.146.1-microsoft

  You may need to install the following packages for this specific kernel:
    linux-tools-5.15.146.1-microsoft-standard-WSL2
    linux-cloud-tools-5.15.146.1-microsoft-standard-WSL2

  You may also want to install one of the following packages to keep up to date:
    linux-tools-standard-WSL2
    linux-cloud-tools-standard-WSL2
```

numactl:

Control and monitor NUMA policy on NUMA systems.

Input: - numactl

Output: -

```
user@LAPTOP-6QVE9SBJ:~$ 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:

Collect and analyze system activity data over time.

Input: - sar -u 1 10

Output: -

```
user@LAPTOP-6QVE9SBJ:~$ sar -u 1 10
Linux 5.15.146.1-microsoft-standard-WSL2 (LAPTOP-6QVE9SBJ)      03/13/24        _x86_64_        (8 CPU)

22:55:43        CPU     %user     %nice   %system   %iowait    %steal     %idle
22:55:44        all      0.38      0.00      0.12      0.00      0.00     99.50
22:55:45        all      0.38      0.00      0.13      0.00      0.00     99.50
22:55:46        all      0.37      0.00      0.62      0.25      0.00     98.75
22:55:47        all      0.25      0.00      0.87      0.00      0.00     98.88
22:55:48        all      0.50      0.00      0.13      0.00      0.00     99.37
22:55:49        all      0.38      0.00      0.25      0.00      0.00     99.38
22:55:50        all      0.25      0.00      0.25      0.00      0.00     99.50
22:55:51        all      0.75      0.00      0.87      0.00      0.00     98.38
22:55:52        all      0.37      0.00      0.62      0.00      0.00     99.00
22:55:53        all      0.50      0.00      0.87      0.00      0.00     98.63
Average:        all      0.41      0.00      0.47      0.02      0.00     99.09
```

Input: - sar -r 60 5

Output: -

```
user@LAPTOP-6QVE9SBJ:~$ sar -r 60 5
Linux 5.15.146.1-microsoft-standard-WSL2 (LAPTOP-6QVE9SBJ)      03/13/24        _x86_64_        (8 CPU)

22:57:04    kbmemfree   kbavail kbmemused  %memused kbbuffers  kbcached  kbcommit   %commit  kbactive   kbinact   kbdirt
y
22:58:04      3137364   3249784    425444     10.81     10660    290496    624372     12.53     88028    393136
0
22:59:04      3122140   3234560    440640     11.19     10660    290496    624768     12.53     88028    403140
0
23:00:04      3124460   3236880    438292     11.13     10660    290496    624372     12.53     88028    404340
0
23:01:04      3116588   3234444    440296     11.19     10800    295468    632768     12.69     89076    409944        19
2
23:02:04      3111532   3229396    445328     11.31     10800    295468    632768     12.69     89076    409980
0
Average:      3122417   3237013    438000     11.13     10716    292485    627810     12.59     88447    404108         3
8
```

Input: - sar -d 5 30

Output: -

```
user@LAPTOP-6QVE9SBJ:~$ sar -d 5 30
Linux 5.15.146.1-microsoft-standard-WSL2 (LAPTOP-6QVE9SBJ)      03/13/24        _x86_64_        (8 CPU)

23:02:32          DEV       tps     rkB/s     wkB/s     dkB/s   areq-sz    aqu-sz     await     %util
23:02:37          sda      0.00      0.00      0.00      0.00      0.00      0.00      0.00      0.00
23:02:37          sdb      0.00      0.00      0.00      0.00      0.00      0.00      0.00      0.00
23:02:37          sdc      0.00      0.00      0.00      0.00      0.00      0.00      0.00      0.00

23:02:37          DEV       tps     rkB/s     wkB/s     dkB/s   areq-sz    aqu-sz     await     %util
23:02:42          sda      0.00      0.00      0.00      0.00      0.00      0.00      0.00      0.00
23:02:42          sdb      0.00      0.00      0.00      0.00      0.00      0.00      0.00      0.00
23:02:42          sdc      0.00      0.00      0.00      0.00      0.00      0.00      0.00      0.00

23:02:42          DEV       tps     rkB/s     wkB/s     dkB/s   areq-sz    aqu-sz     await     %util
23:02:47          sda      0.00      0.00      0.00      0.00      0.00      0.00      0.00      0.00
23:02:47          sdb      0.00      0.00      0.00      0.00      0.00      0.00      0.00      0.00
23:02:47          sdc      0.00      0.00      0.00      0.00      0.00      0.00      0.00      0.00

23:02:47          DEV       tps     rkB/s     wkB/s     dkB/s   areq-sz    aqu-sz     await     %util
23:02:52          sda      0.00      0.00      0.00      0.00      0.00      0.00      0.00      0.00
23:02:52          sdb      0.00      0.00      0.00      0.00      0.00      0.00      0.00      0.00
23:02:52          sdc      0.00      0.00      0.00      0.00      0.00      0.00      0.00      0.00

23:02:52          DEV       tps     rkB/s     wkB/s     dkB/s   areq-sz    aqu-sz     await     %util
23:02:57          sda      0.00      0.00      0.00      0.00      0.00      0.00      0.00      0.00
23:02:57          sdb      0.00      0.00      0.00      0.00      0.00      0.00      0.00      0.00
23:02:57          sdc      0.00      0.00      0.00      0.00      0.00      0.00      0.00      0.00
```

Image Gray scaling

Abhay Magar(23MDS001)
Vaibhav Raval(23MDS010)
Aashita Jain(23MDS015)

Types of Images:

1. Color Image

2. Grayscale Image

3. Binary Image

(a) (b) (c)

Gray Scale Image:

1. A gray-scale (or gray level) image is simply one in which the only colors are shades of gray. Unlike color images, which contain multiple color channels (typically red, green, and blue), grayscale images contain only one channel representing the intensity of light at each pixel.
2. In a gray-scale image, each pixel has a value between 0 and 255, where zero corresponds to “black” and 255 corresponds to “white”.
3. The values between 0 and 255 are varying shades of gray, where values closer to 0 are darker and values closer to 255 are lighter.
4. For RGB (Red, Green, Blue) color images, typically each color channel (Red, Green, Blue) is represented using 8 bits, giving a total of 24 bits for the three channels combined. Grayscale images typically use only one channel to represent the intensity of light at each pixel. Hence, it is represented only using 8-bit.

Why is grayscale needed for image processing?

1. It helps in simplifying algorithms and as well eliminates the complexities related to computational requirements.
2. It makes room for easier learning for those who are new to image processing.
3. It enhances easy visualization. It differentiates between the shadow details and the highlights of an image because it is mainly in 2 spatial dimensions (2D) rather than 3D.
4. Color complexity is also reduced.

Methods available to convert an RGB image to grayscale

1. Luminosity Method:

   $$I = 0.21 * R + 0.72 * G + 0.07 * B$$

2. Average Method:

   The grayscale value of each pixel is computed as the average of its red, green, and blue components:

   $$I = (R + G + B) / 3$$

3. Single channel extraction:

   Simply take one of the RGB channels (usually green, as the human eye is most sensitive to green) as the grayscale image.

Luminosity Method

```
import numpy as np
import cv2

def luminosity_method(img):
    # Extracting the Red, Green, and Blue channels
    R = img[:,:,0]
    G = img[:,:,1]
    B = img[:,:,2]

    # Applying the luminosity method formula
    grayscale_img = 0.21 * R + 0.72 * G + 0.07 * B

    # Convert the grayscale image to uint8 datatype
    grayscale_img = np.uint8(grayscale_img)

    return grayscale_img

color_img = cv2.imread('flower1.jpg')

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

# Display the original and grayscale images
cv2.imshow('Color Image', color_img)
cv2.imshow('Grayscale Image (Luminosity Method)', grayscale_img_lum)
```

Average Method:

```
import numpy as np
import cv2

def average_method(img):
    # Calculate the average of RGB channels
    grayscale_img = np.mean(img, axis=2)

    # Convert the grayscale image to uint8 datatype
    grayscale_img = np.uint8(grayscale_img)

    return grayscale_img

# Read the color image
color_img = cv2.imread('animated1.jpg')

# Convert the color image to grayscale using the average method
grayscale_img_avg = average_method(color_img)

# Display the original and grayscale images
cv2.imshow('Color Image', color_img)
cv2.imshow('Grayscale Image (Average Method)', grayscale_img_avg)
```

Single Channel Method

```
import cv2

# Read the color image
color_img = cv2.imread('flower.jpg')

green_channel = color_img[:,:,1]

# Extract the red channel
red_channel = color_img[:,:,2]

# Extract the blue channel
blue_channel = color_img[:,:,0]

# Display the original and grayscale images
cv2.imshow('Color Image', color_img)
cv2.imshow('Grayscale Image (Single Channel - Green)', green_channel)
cv2.imshow('Grayscale Image (Single Channel - Red)', red_channel)
cv2.imshow('Grayscale Image (Single Channel - Blue)', blue_channel)
```

Color Image

Green Channel

Red Channel

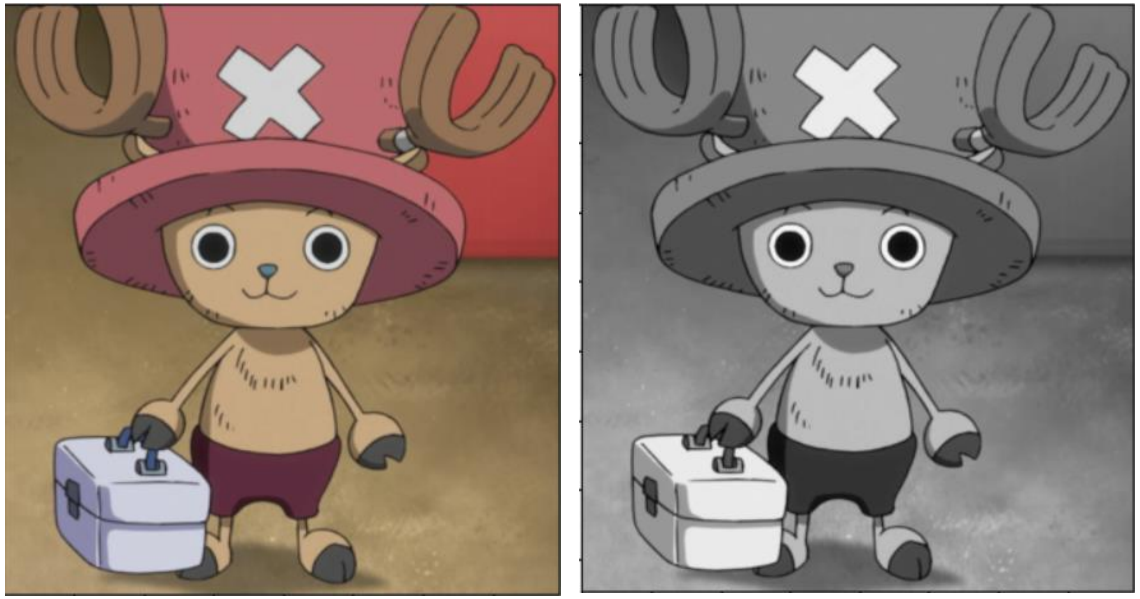
Blue Channel

Pillow:

```
from PIL import Image
img = Image.open('toji.webp').convert('L')
img.save('greyscale.png')
```

Using Matplotlib and Scikit Learn

```
import matplotlib.pyplot as plt
from skimage import io
from skimage import data
from skimage.color import rgb2gray

rgb_img  = io.imread('chopper.webp’)

gray_img  = rgb2gray(rgb_img)
fig, axes = plt.subplots(1, 2, figsize=(8, 4))
ax = axes.ravel()

ax[0].imshow(rgb_img)
ax[0].set_title("Original image")
ax[1].imshow(gray_img, cmap=plt.cm.gray)
ax[1].set_title("Processed image")

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



Using cv2

```
import cv2

# Load the input image
image = cv2.imread('tanjiro.jfif')
cv2.imshow('Original', image)
cv2.waitKey(0)

# Use the cvtColor() function to grayscale the image
gray_image = cv2.cvtColor(image,
cv2.COLOR_BGR2GRAY)

cv2.imshow('Grayscale', gray_image)
cv2.waitKey(0)

# Window shown waits for any key pressing event
cv2.destroyAllWindows()
```

Applications of Grayscale Images:

- **Medical Imaging:** Grayscale images are extensively used in medical imaging for tasks such as X-rays, MRIs, CT scans, and ultrasound imaging.

- **Document Processing:** Grayscale images are often used in document processing applications such as scanning and OCR (Optical Character Recognition). They can help in extracting text and important features from documents.

- **Image Processing and Computer Vision:** Grayscale images are widely used in image processing and computer vision tasks such as object detection, image segmentation, object recognition, and feature extraction. Their simplicity makes them computationally less expensive to work with compared to full-color images.

THANK YOU

Image Filtering

In [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
```

```
C:\Users\raval\anaconda3\Lib\site-packages\paramiko\transport.py:219: CryptographyDeprecat
ionWarning: Blowfish has been deprecated
  "class": algorithms.Blowfish,
```

In [2]:
```
original_image = cv2.imread(r"C:\Users\raval\Downloads\images\images\Lenna_(test_image).png
```

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

Image nosing

In [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.ui
        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[
        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.uin
        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_gener
```

In [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()
```

Linear Filtering

Box Filter

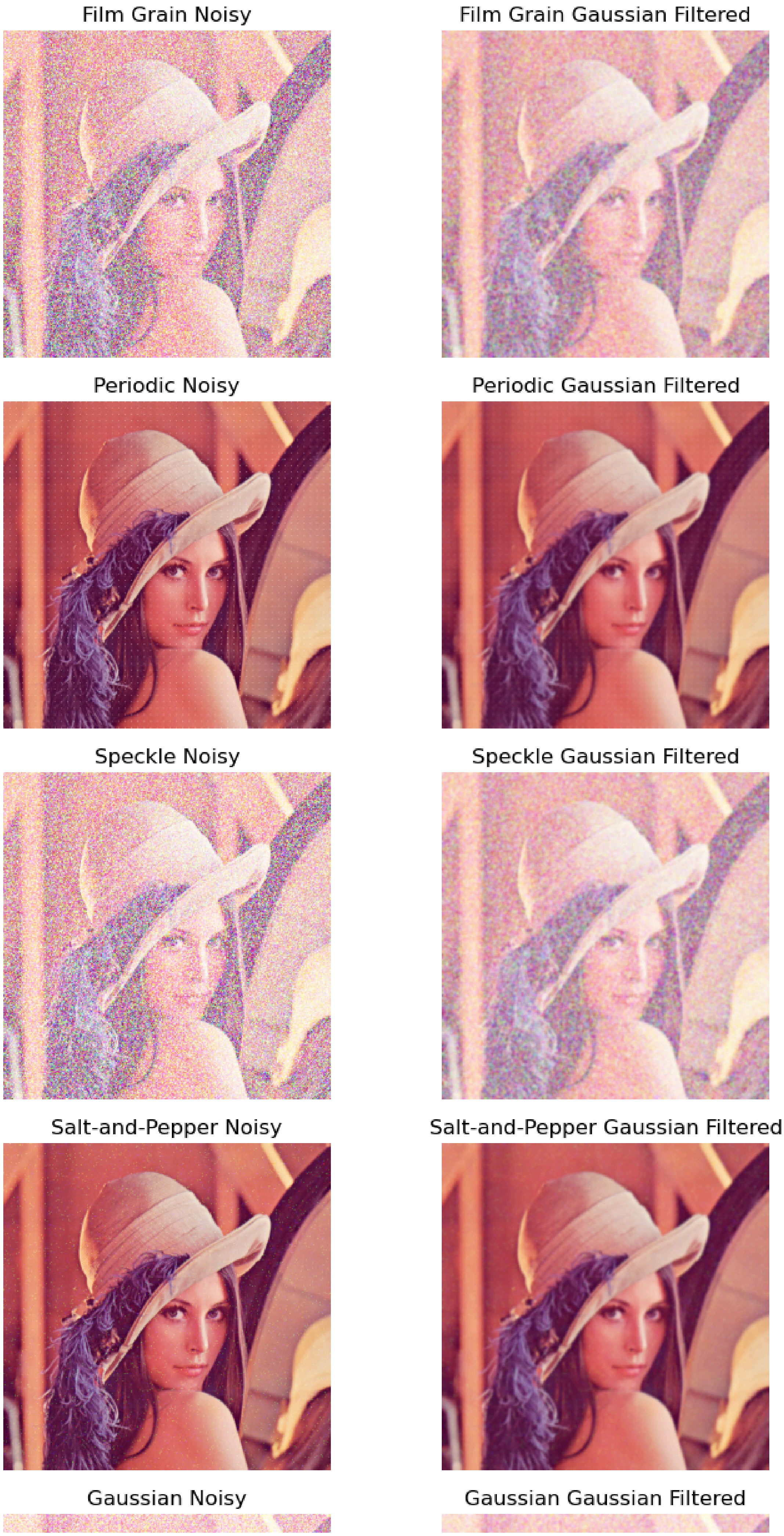
In [6]:
```
def apply_box_filter(image):
    return cv2.boxFilter(image, -1, (5, 5))

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

Gaussian Filter

In [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

Sobel

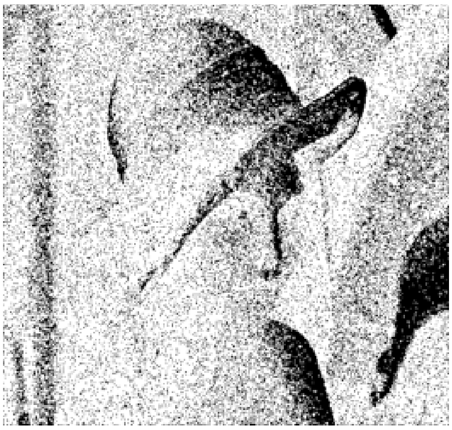
In [8]:
```
if original_image is None:
    print("Error: Unable to load the image.")
else:
    sobel_x = cv2.Sobel(original_image, cv2.CV_64F, 1, 0, ksize=3)
    sobel_y = cv2.Sobel(original_image, cv2.CV_64F, 0, 1, ksize=3)

    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

In [9]:
```python
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
Speckle Edges
Salt-and-Pepper Noisy
Salt-and-Pepper Edges
Gaussian Noisy
Gaussian Edges



Non-linear Filtering

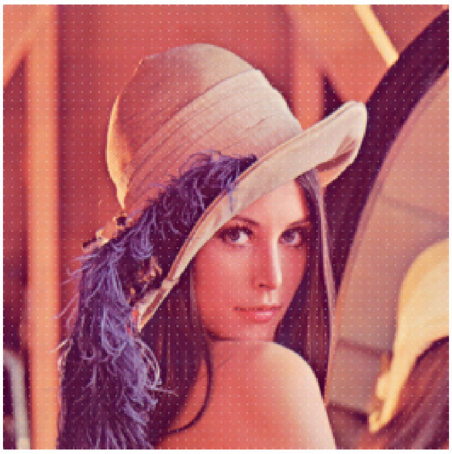
Median Filter

In [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

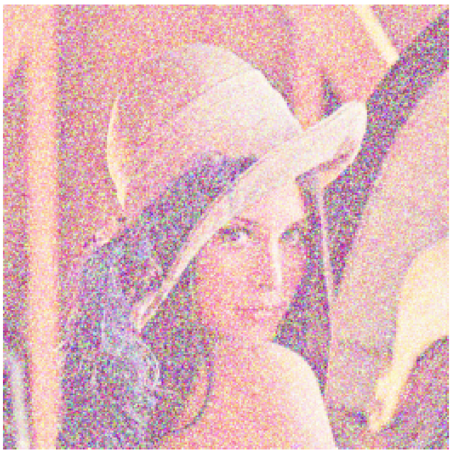
Film Grain Median Filtered

Periodic Noisy



Periodic Median Filtered

Speckle Noisy



Speckle Median Filtered

Salt-and-Pepper Noisy

Salt-and-Pepper Median Filtered

Gaussian Noisy

Gaussian Median Filtered

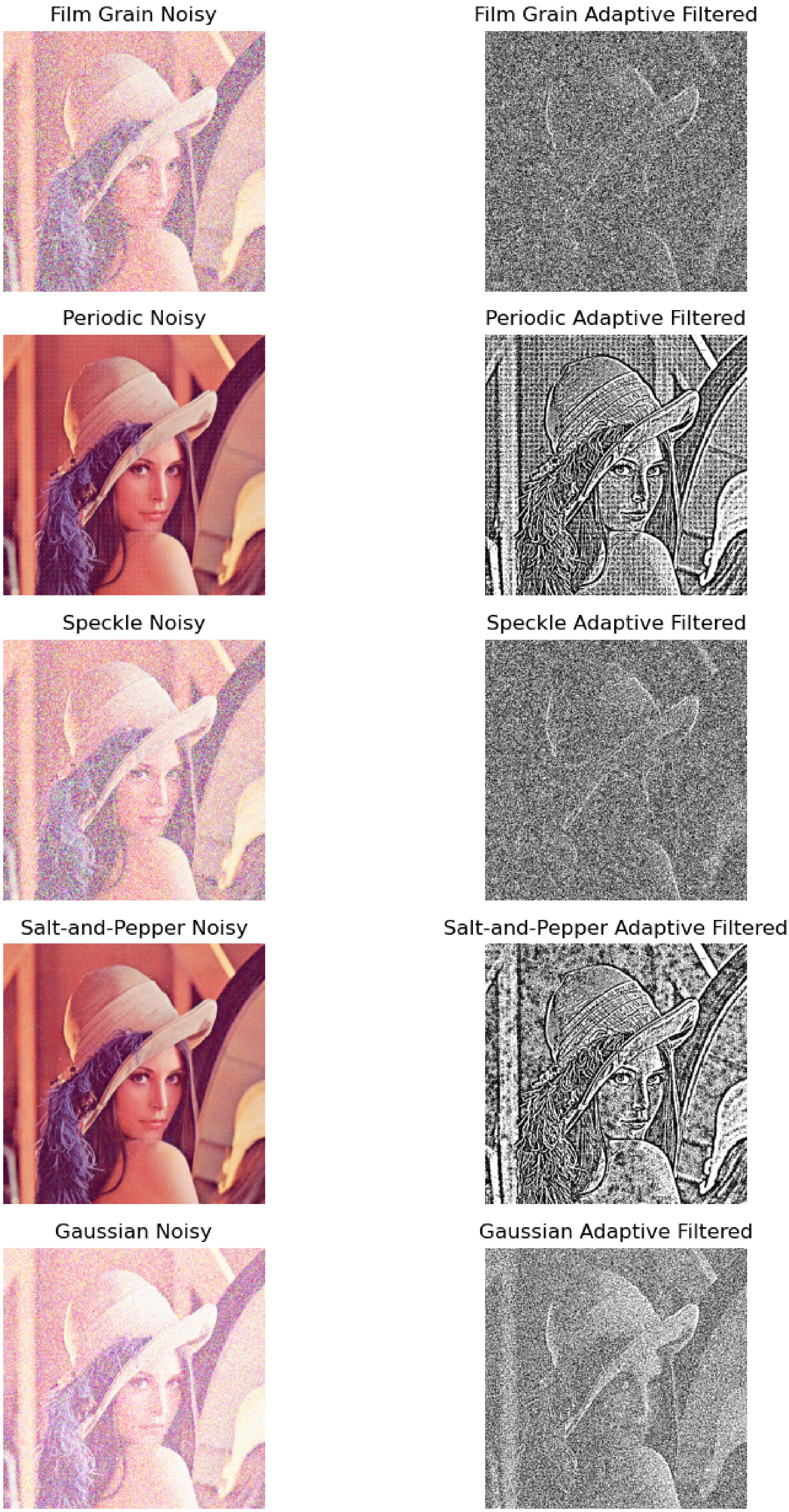
Rank Filter

In [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

Adaptive Filter

In [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.THRES

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

Bilateral Filter

In [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

Image Filtering on colour image

In [14]:
```
bgr_image = cv2.imread(r"C:\Users\raval\Downloads\images\images\flower.jpg")
original_image = cv2.cvtColor(bgr_image, cv2.COLOR_BGR2RGB)
```

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

In [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.uin
        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[
        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_gener
```

In [17]:

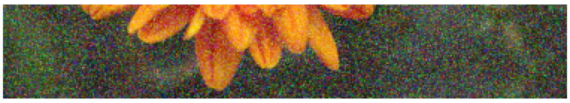
```
plt.figure(figsize=(15, 15))
for i, (noise_type, noisy_image) in enumerate(noisy_images.items(), start=1):
        plt.subplot(3, 2, i)
        plt.imshow(noisy_image)
        plt.title(noise_type)
        plt.axis('off')
plt.show()
```

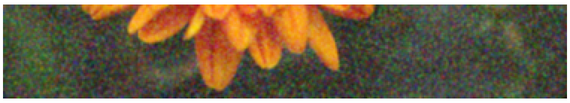
Linear Filtering

BOX Filter

In [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

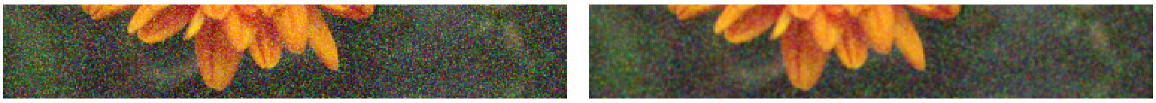




Gaussian Filter

In [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()
```

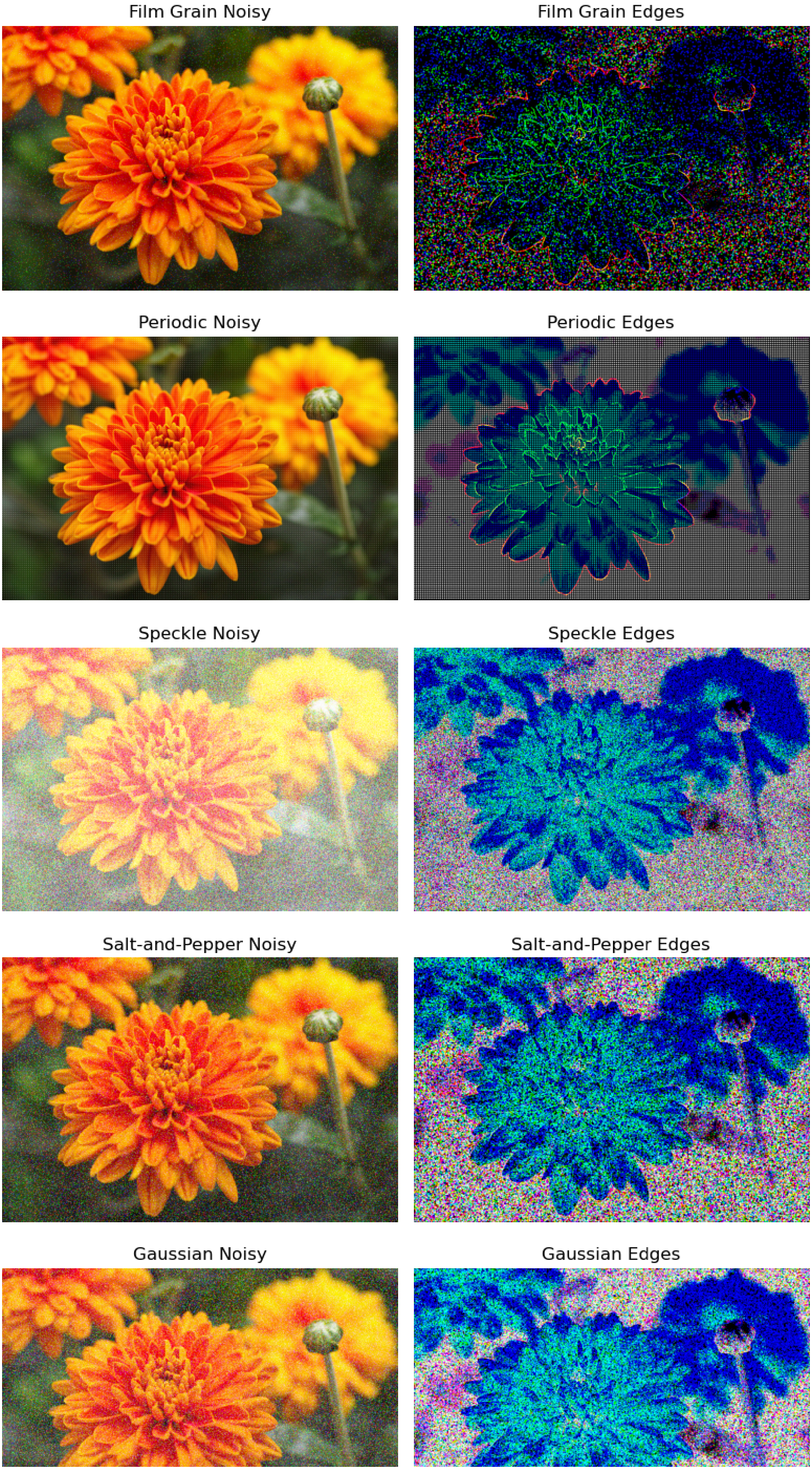

Film Grain Noisy
Film Grain Gaussian Filtered
Periodic Noisy
Periodic Gaussian Filtered
Speckle Noisy
Speckle Gaussian Filtered
Salt-and-Pepper Noisy
Salt-and-Pepper Gaussian Filtered
Gaussian Noisy
Gaussian Gaussian Filtered

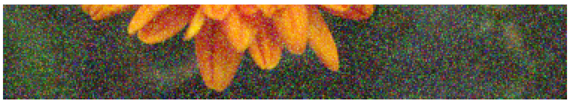
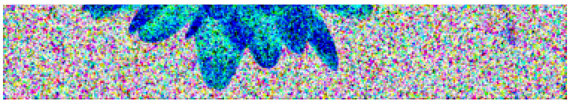


Sobel

In [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()
```

In [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
Speckle Edges
Salt-and-Pepper Noisy
Salt-and-Pepper Edges
Gaussian Noisy
Gaussian Edges

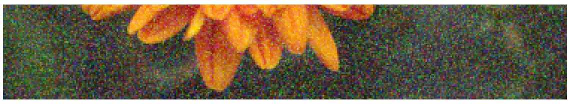
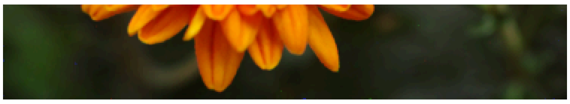



Non-linear Filtering

Median Filter

In [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()
```

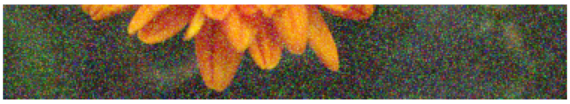

Film Grain Noisy
Film Grain Median Filtered
Periodic Noisy
Periodic Median Filtered
Speckle Noisy
Speckle Median Filtered
Salt-and-Pepper Noisy
Salt-and-Pepper Median Filtered
Gaussian Noisy
Gaussian Median Filtered

Rank Filter

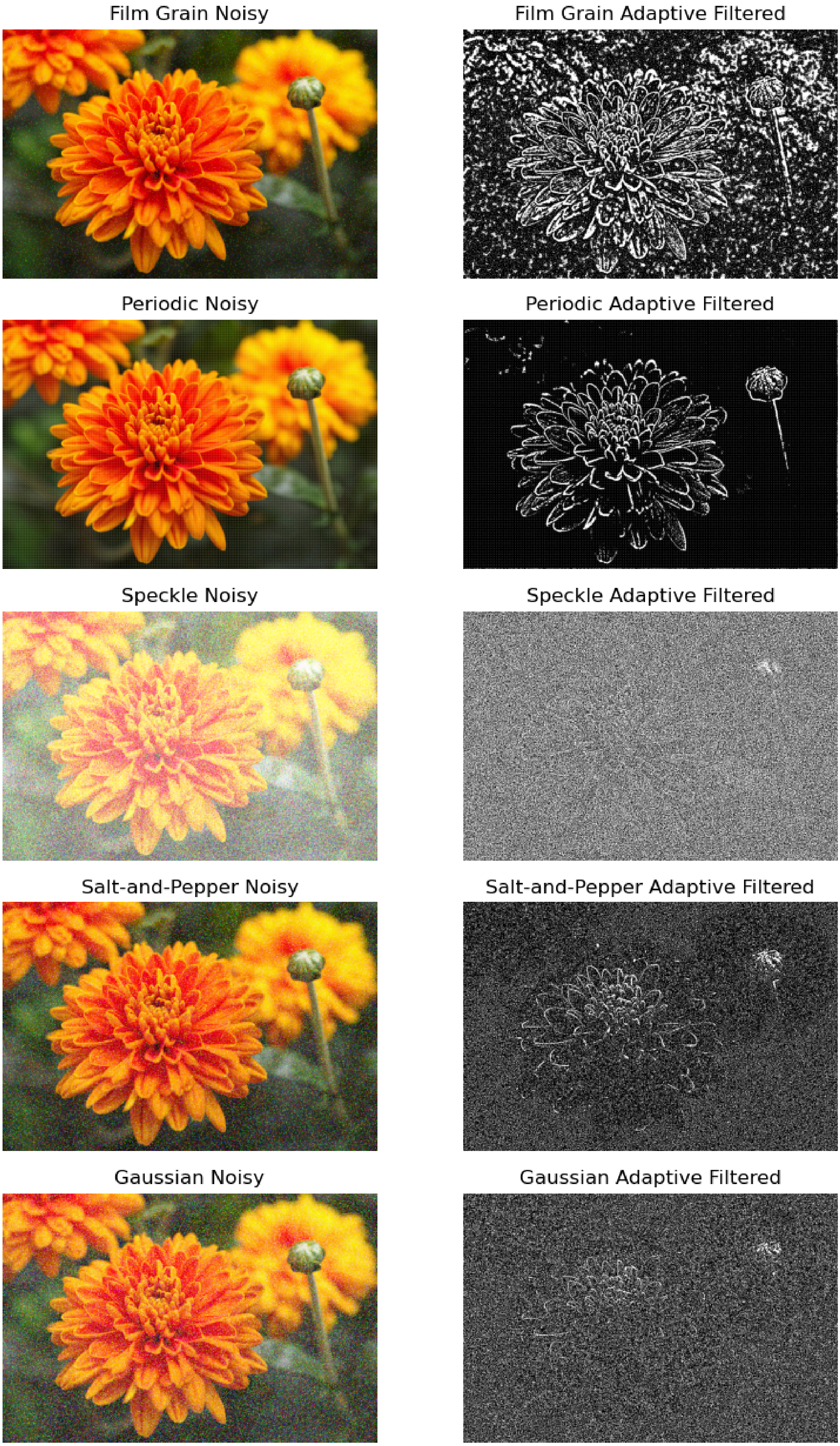
In [23]:
```
def apply_rank_filter(image):
    kernel_size = 3
    rank = 4

    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



Adaptive Filter

In [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.THRES

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

Bilateral Filter

In [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

Gray Scale Imaging

Average Method

In [27]:
```
def average_method(img):
    grayscale_img = np.mean(img, axis=2)

    grayscale_img = np.uint8(grayscale_img)

    return grayscale_img

color_img = cv2.imread(r"C:\Users\raval\Downloads\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()
```

Color Image

Grayscale Image (Average Method)

Luminosity Method

In [29]:
```
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(r"C:\Users\raval\Downloads\images\images\flower1.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()
```

Color Image

Grayscale Image (Luminosity Method)

Single Channel Method

In [31]:
```
color_img = cv2.imread(r"C:\Users\raval\Downloads\images\images\flower1.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)

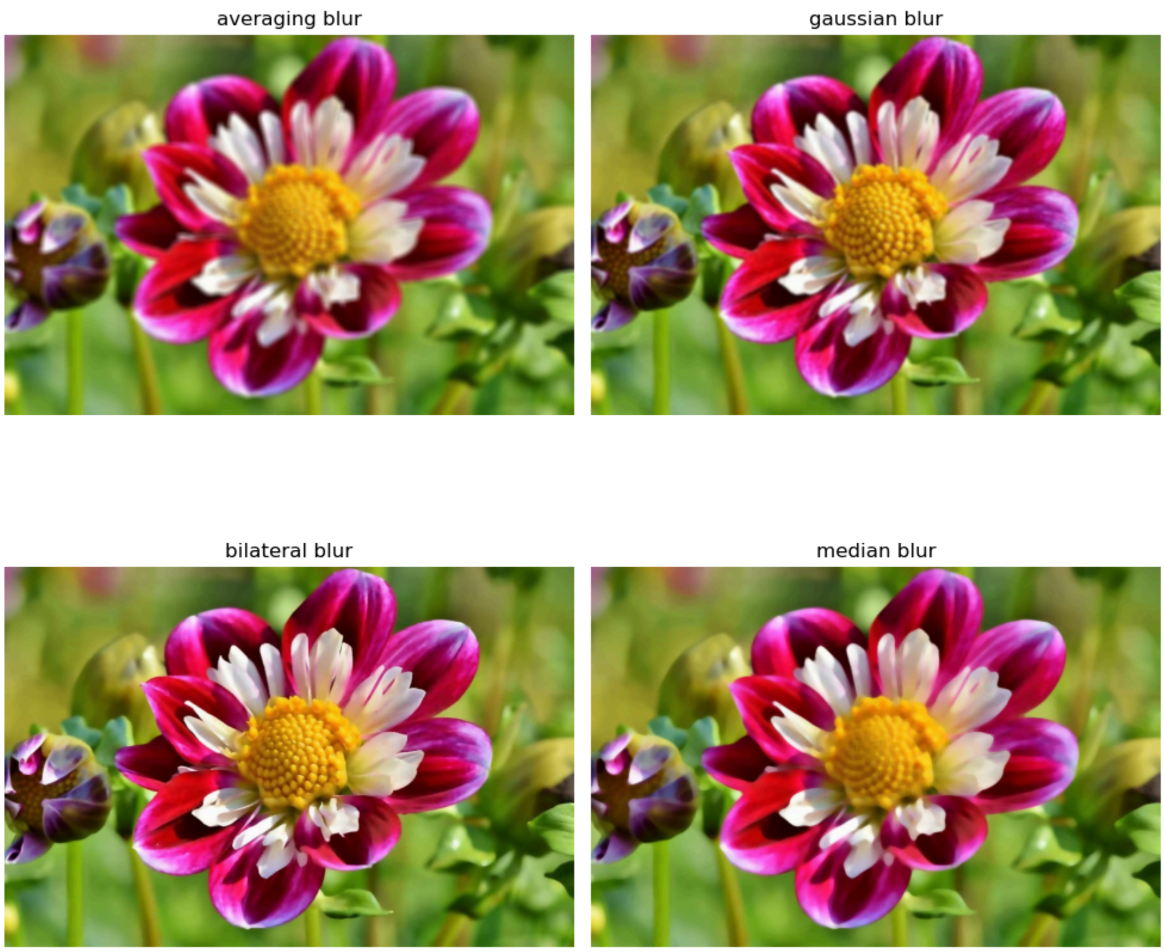
Image Blurring

In [32]:
```
image_path = r"C:\Users\raval\Downloads\images\images\flower1.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()
```

In [33]:
```
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)
```

In [34]:
```
display_size = (200, 200)
original_display = cv2.resize(img, display_size)
```

In [35]:
```
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()
```



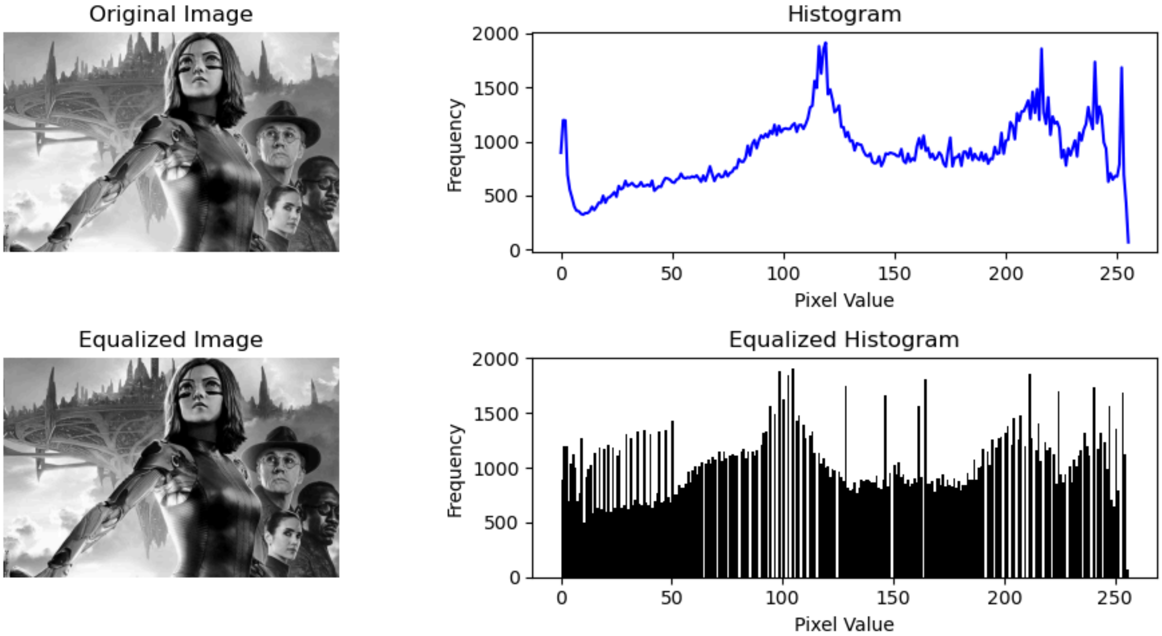
Histogram based image analysis

In [36]:
```
image_path = r"C:\Users\raval\Downloads\images\images\animated1.jpg"
image = Image.open(image_path).convert('L')
```

In [37]:
```
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
```

In [38]:
```
equalized_image, hist = histogram_equalization(np.array(image))
```

In [39]:
```
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()
```



In [40]:
```
image = Image.open(r"C:\Users\raval\Downloads\images\images\Fig0310(b)(washed_out_pollen_im
```

In [41]:
```
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
```

In [42]:
```
equalized_image, hist_original, cdf_equalized = histogram_equalization(np.array(image))
```

In [43]:

```
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()
```

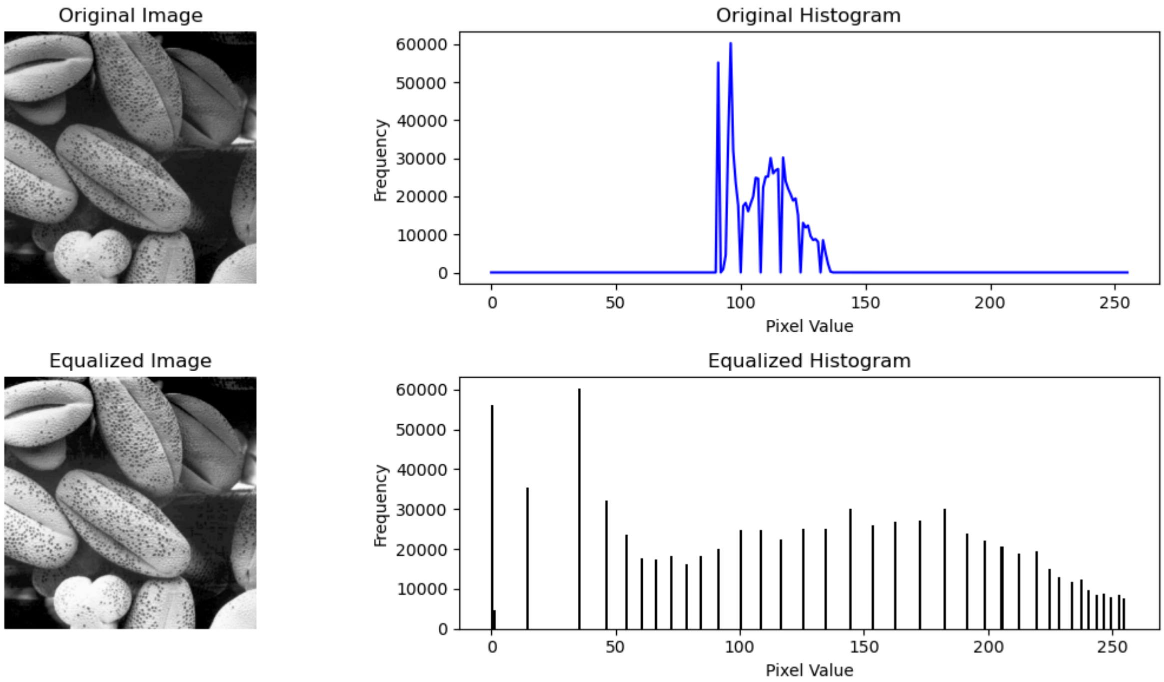


Image Scaling

Nearest neighbor

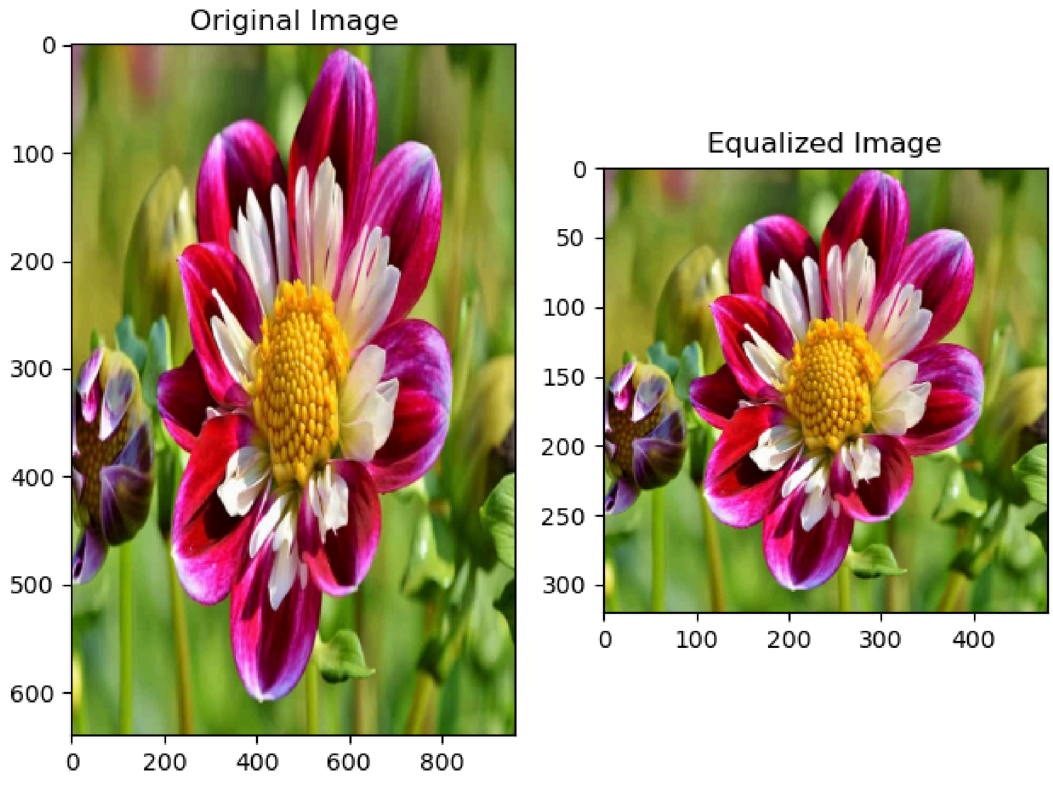
In [44]:

```
from PIL import Image, ImageDraw, ImageFont
from IPython.display import display
```

In [45]:
```python
original_img = Image.open(r"C:\Users\raval\Downloads\images\images\flower1.jpg")

original_width, original_height = original_img.size

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

In [46]:
```python
resized_img = original_img.resize((new_width, new_height), resample=Image.NEAREST)
```

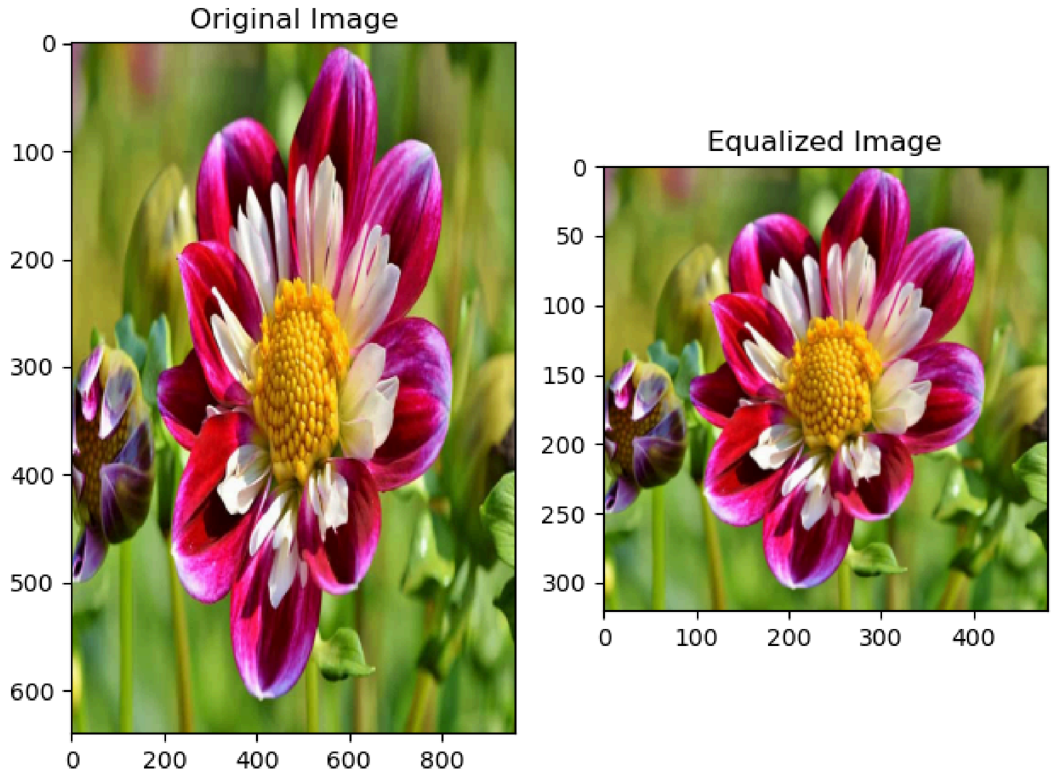
In [47]:
```python
original_width, original_height = original_img.size

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

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



Bilinar Interpolation

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

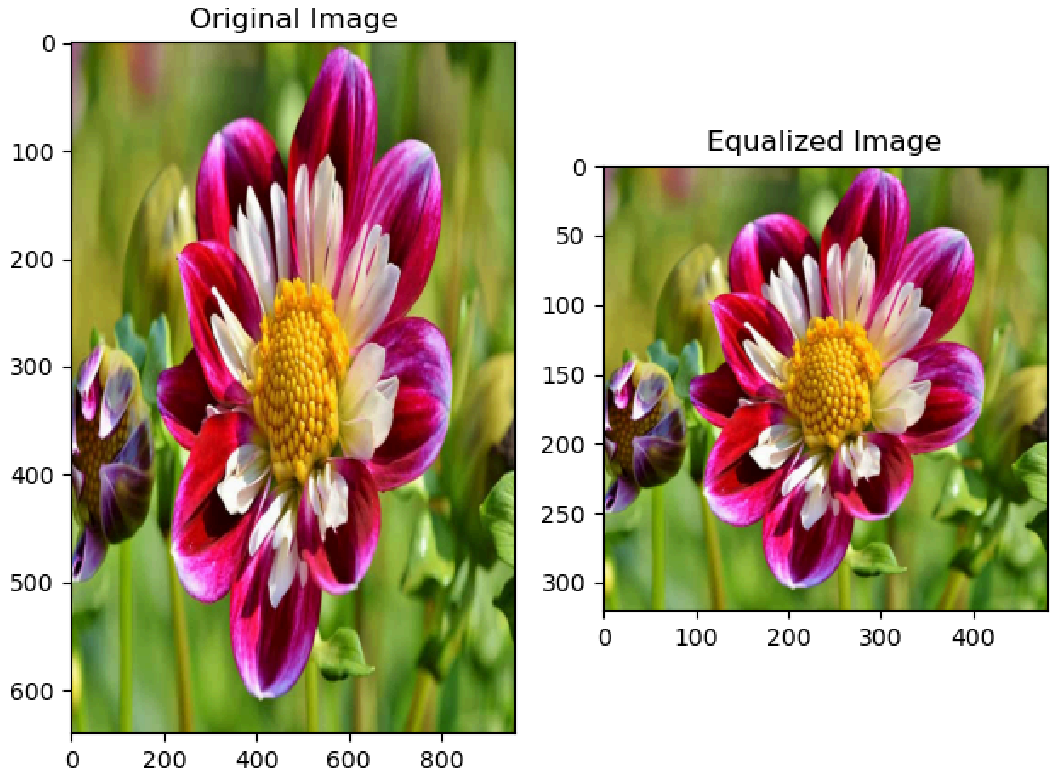
In [49]:
```
original_width, original_height = original_img.size

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

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



Bicubic Interpolation

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

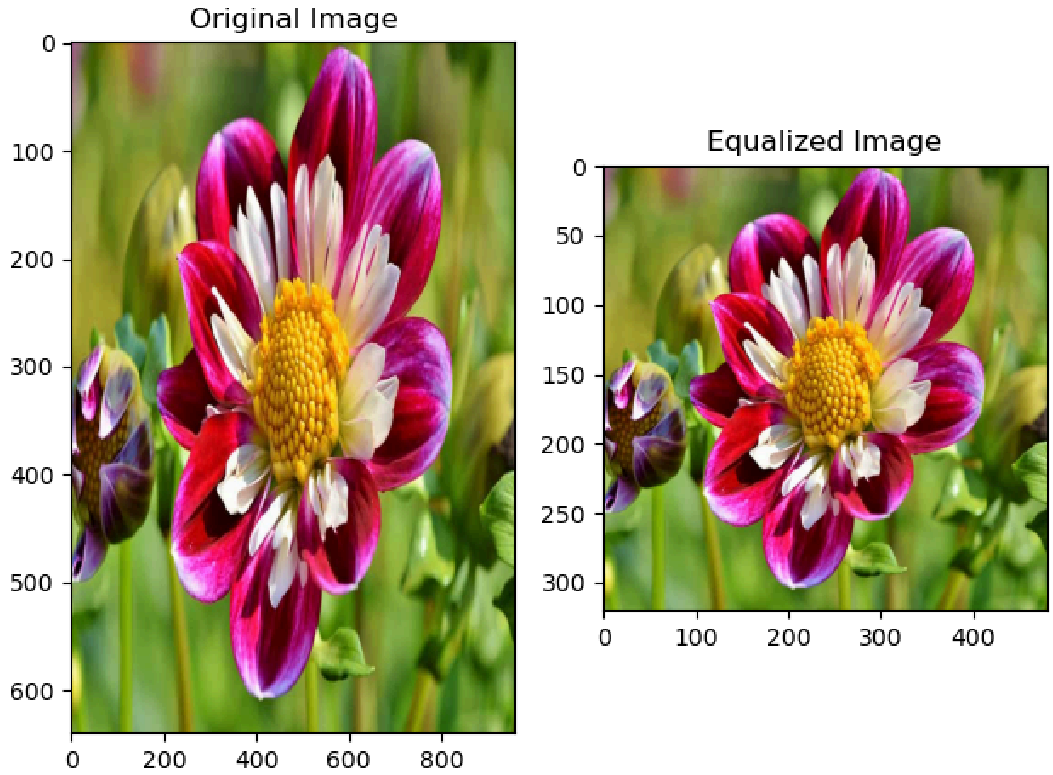
In [51]:
```
original_width, original_height = original_img.size

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

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



Lanczos Interpolation

In [52]:
```
resized_img = original_img.resize((new_width, new_height), resample=Image.LANCZOS)
```

In [53]:
```
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

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



In [ ]:

Assignment-16

1. What is CUDA?

   CUDA stands for Compute Unified Device Architecture. It is a parallel computing platform and programming model developed by NVIDIA for general computing on graphical processing units (GPUs). It allows developers to harness the computational power of NVIDIA GPUs for parallel processing tasks, such as scientific simulations, image processing, deep learning, and more.

2. What is the prerequisite for learning CUDA?

   Strong understanding of parallel computing concepts, familiarity with C or C++ programming languages, and basic knowledge of GPU architecture and memory hierarchy.

3. Which are the languages that support CUDA?

   CUDA primarily supports C and C++ programming languages. However, there are also libraries and frameworks available for other languages like Python (with libraries like PyCUDA or Numba), Fortran, and even MATLAB for CUDA programming.

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

   CUDA-ready architecture refers to GPUs that are designed and optimized to work efficiently with CUDA programming. These GPUs have specialized hardware components and software support to accelerate parallel processing tasks performed using CUDA.

5. How CUDA works?

   Big Task: Imagine you have a huge math problem to solve.

Ask for Help: Instead of doing it alone, you ask your GPU friend for help.

Divide the Work: Your GPU friend splits the math problem into smaller parts.

Everyone Works Together: Each part is solved at the same time by different parts of the GPU.

Faster Solution: Because everyone is working together, the math problem gets solved much faster than if you did it alone.

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

   Benefits:

   Accelerated performance for parallelizable tasks.

   Utilization of GPU resources for computational tasks.

   Scalability across different CUDA-enabled GPUs.

   Support for a wide range of parallel computing applications.

   Limitations:

   Requires a dedicated NVIDIA GPU.

   Complexity in managing memory hierarchy.

   Limited support for non-NVIDIA GPUs.

   Requires understanding of parallel programming concepts.

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

   Include Necessary Headers:

   #include <iostream>: Include the standard input/output stream library for printing results.

   Define CUDA Kernel Function:

   global void vectorAdd(int *a, int *b, int *c, int n): Define the CUDA kernel function vectorAdd, which adds corresponding elements of input vectors a and b and stores the result in the output vector c. This function will be executed on the GPU.

   Main Function:

   int main(): Define the main function.

   Vector Size and Host Vectors:

   int n = 1024;: Define the size of the vectors.

int *h_a, *h_b, *h_c;: Declare host pointers for input vectors a, b, and the output vector c.

h_a = new int[n];, h_b = new int[n];, h_c = new int[n];: Allocate memory on the host for vectors a, b, and c.

Initialize Input Vectors:

for (int i = 0; i < n; i++) { h_a[i] = i; h_b[i] = i * 2; }: Initialize input vectors a and b with some values.

Allocate Device Memory:

int *d_a, *d_b, *d_c;: Declare device pointers for vectors a, b, and c.

cudaMalloc((void*)&d_a, n * sizeof(int));, cudaMalloc((void)&d_b, n * sizeof(int));, cudaMalloc((void*)&d_c, n * sizeof(int));: Allocate memory on the GPU for vectors a, b, and c.

Copy Input Data from Host to Device:

cudaMemcpy(d_a, h_a, n * sizeof(int), cudaMemcpyHostToDevice);: Copy input vector a from host to GPU.

cudaMemcpy(d_b, h_b, n * sizeof(int), cudaMemcpyHostToDevice);: Copy input vector b from host to GPU.

Launch CUDA Kernel:

int threadsPerBlock = 256;: Define the number of threads per block.

int blocksPerGrid = (n + threadsPerBlock - 1) / threadsPerBlock;: Calculate the number of blocks needed in the grid.

vectorAdd<<<blocksPerGrid, threadsPerBlock>>>(d_a, d_b, d_c, n);: Launch the vectorAdd CUDA kernel on the GPU.

Copy Output Data from Device to Host:

cudaMemcpy(h_c, d_c, n * sizeof(int), cudaMemcpyDeviceToHost);: Copy the output vector c from GPU to host.

Print Results:

for (int i = 0; i < 10; i++) { std::cout << h_c[i] << " "; }: Print the first 10 elements of vector c.

Free Device and Host Memory:

cudaFree(d_a);, cudaFree(d_b);, cudaFree(d_c);: Free memory allocated for vectors a, b, and c on the GPU.

delete[] h_a;, delete[] h_b;, delete[] h_c;: Free memory allocated for vectors a, b, and c on the host.

Return 0:

return 0;: Indicate successful program completion

8. Explain CUDA thread organization.

CUDA thread organization refers to how threads are arranged and managed within CUDA programs for execution on the GPU. Understanding CUDA thread organization is crucial for efficiently utilizing the computational power of the GPU. Here are the key aspects of CUDA thread organization:

Threads: Threads are the smallest units of execution in CUDA programs. Each thread executes a specific portion of the overall computation.

Thread Hierarchy:

Grid: A grid is the top-level organization of threads in a CUDA program. It consists of multiple thread blocks.

Thread Block: A thread block is a group of threads that can cooperate with each other via shared memory and synchronization. All threads within a block have access to the same resources and can synchronize with each other.

Threads: Threads within a block are organized in one, two, or three dimensions, forming a thread block grid. Each thread has a unique identifier within its block.

Thread Indexing:

Thread Index: Each thread is identified by its unique index within its block. In a one-dimensional block grid, threads are indexed from 0 to (blockDim.x - 1). Similarly, in two or three-dimensional block grids, threads are indexed in a multidimensional manner.

Block Index: Each block is identified by its unique index within the grid. In a one-dimensional grid, blocks are indexed from 0 to (gridDim.x - 1). In multidimensional grids, blocks are indexed accordingly.

Global Thread Index: Threads are often referenced using their global thread index, which is a unique combination of block index and thread index within the grid. This index allows threads to access unique portions of data or perform specific computations based on their position within the grid.

Thread Synchronization:

Threads within the same block can synchronize their execution using synchronization primitives such as barriers (__syncthreads()). Synchronization ensures that all threads have finished executing certain instructions before proceeding.

Threads from different blocks cannot directly synchronize with each other.

Memory Hierarchy:

Each thread has access to different types of memory:

Registers: Each thread has its own set of registers for storing temporary data.

Shared Memory: Threads within the same block can share data through shared memory, which has low latency and is visible to all threads within the block.

Global Memory: All threads in the grid have access to global memory, which is the main memory space on the GPU. Accessing global memory is slower compared to shared memory.

Local Memory: Local memory is used for storing local variables and is slower than shared memory.

Thread Divergence:

Thread divergence occurs when different threads within the same block take different execution paths based on conditional statements. This can lead to inefficiencies due to serialization of execution.

Overall, understanding CUDA thread organization allows developers to effectively utilize the parallelism offered by GPUs and optimize their CUDA programs for performance.

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

Not able to install CUDA in local system due to unavailability of NVIDIA GPU.

Assignment 17

Implement following CUDA programs

1. to print hello message on the screen using kernal function
2. to add two vectors of size 100 and 20000 abd analyze the performance comparison between cpu and gpu processing
3. to multply two matrix of size 20 X 20 and 1024 X 1024 analyze the performance comparison between cpu and gpu processing
4. to obtain CUDA device information and print the output

```
!pip install pycuda
```

```
Collecting pycuda
  Using cached pycuda-2024.1-cp310-cp310-linux_x86_64.whl
Collecting pytools>=2011.2 (from pycuda)
  Using cached pytools-2024.1.1-py2.py3-none-any.whl (85 kB)
Requirement already satisfied: appdirs>=1.4.0 in /usr/local/lib/python3.10/dist-packages (from pycuda) (1.4.4)
Collecting mako (from pycuda)
  Using cached Mako-1.3.3-py3-none-any.whl (78 kB)
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)
Installing collected packages: pytools, mako, pycuda
Successfully installed mako-1.3.3 pycuda-2024.1 pytools-2024.1.1
```

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;
}
```

```
Overwriting 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)!
```

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

GPU

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

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

```
Vector addition of size 100 on GPU took 0.000827789306640625 seconds.
Vector addition of size 20000 on GPU took 8.726119995117188e-05 seconds.
```

CPU

```
def vector_add_cpu(a, b):
    start_time = time.time()
    result = a + b
    end_time = time.time()
    return result, 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_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])
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 2.2649765014648438e-05 seconds.
Vector addition of size 20000 on CPU took 1.621246337890625e-05 seconds.
```

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

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: 0.434800 seconds

Matrix size: (1024, 1024)
GPU time: 0.013687 seconds
```

CPU

```
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

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

def measure_time(matrix_size, func, *args):
```

4. To obtain CUDA device information and print the output

```
import pycuda.driver as cuda

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
```

```
!nvidia-smi
```

```
Tue Apr 16 10:28:52 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   75C    P0              32W /  70W |    103MiB / 15360MiB |      0%      Default |
|                                         |                      |                  N/A |
+-----------------------------------------+----------------------+----------------------+

+---------------------------------------------------------------------------------------+
| Processes:                                                                            |
|  GPU   GI   CI        PID   Type   Process name                            GPU Memory |
|        ID   ID                                                             Usage      |
|=======================================================================================|
+---------------------------------------------------------------------------------------+
```

Start coding or generate with AI.

Lab Assignment 18

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

Gaussian Blur

Description

Gaussian blur is a widely used image processing operation that helps in reducing image noise and details, thus creating a smoother image. It works by averaging the intensity of pixels in the vicinity of each pixel, weighted by a Gaussian distribution. This weighted averaging process blurs the image while preserving its overall structure.

Parallelism Insertion

1. Divide the Workload: Split the image processing tasks among multiple threads, each handling a portion of the image.
2. Use GPU-accelerated Operations: Leverage CuPy's GPU-accelerated functions to perform image processing operations on the GPU.
3. Parallel Kernel Launch: Launch a CUDA kernel with multiple threads to execute the processing tasks concurrently on the GPU.
4. Ensure Synchronization: Synchronize the GPU to ensure all threads have completed their tasks before proceeding to the next steps or accessing the processed data.
5. Optimize Memory Usage: Utilize GPU memory efficiently by minimizing data transfers between the CPU and GPU and optimizing memory allocation and deallocation

Performance Analysis

Sequential

```
import numpy as np
from scipy.signal import convolve2d
from PIL import Image
import os
import time

def process_image(image_array):
    def gaussian_kernel(size, sigma=1):
        kernel_1D = np.linspace(-(size // 2), size // 2, size)
        for i in range(size):
            kernel_1D[i] = np.exp(-0.5 * (kernel_1D[i] / sigma) ** 2)
        kernel_2D = np.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 = convolve2d(image_array, gaussian_kernel_array, mode='same', boundary='wrap')

    return blurred_image

directory = "/content/drive/MyDrive/NNDL/dog cat/dog cat/train/cats"
num_images = 0
start_time = time.time()

image_paths = [os.path.join(directory, filename) for filename in os.listdir(directory) if filename.endswith(".jpg")]
num_images = len(image_paths)

for image_path in image_paths:
    image_array = np.array(Image.open(image_path).convert("L"))
    image_blurred = process_image(image_array)

total_time_sequential = time.time() - start_time

print("Number of images processed in sequence:", num_images)
print("Time taken for sequential processing:", total_time_sequential, "seconds")
```

```
Number of images processed in sequence: 279
Time taken for sequential processing: 38.25399899482727 seconds
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

Parallel

```
import cupy 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

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: 279
Time taken for parallel processing: 2.9992127418518066 seconds
```

FFT - Fast Fourier Transform

Description

The Fast Fourier Transform (FFT) is a widely used algorithm for efficiently computing the Discrete. It transforms a signal from its time or spatial domain into its frequency domain, revealing the frequency components present in the signal. FFT has numerous applications in signal processing, image processing, data compression, and more

Parallelism Insertion

1. Divide and Conquer: Divide the input data into smaller chunks and distribute them among multiple threads on the GPU.
2. Utilize GPU-accelerated Libraries: Leverage GPU-accelerated libraries like CuPy, which provide efficient implementations of FFT algorithms optimized for GPU execution.
3. Parallel Kernel Launch: Launch a CUDA kernel with multiple threads to perform parallel FFT computation on the GPU. Each thread processes a portion of the input data independently.
4. Ensure Synchronization: Synchronize the GPU to ensure all threads have completed their FFT computations before proceeding to the next steps or accessing the results.
5. Optimize Memory Usage: Optimize memory access patterns and data transfers between the CPU and GPU to minimize overhead and maximize throughput.

Performance Analysis

Sequential

```
import numpy as np
import matplotlib.pyplot as plt
from scipy import fftpack
from PIL import Image
import os
import time

num_images = 0
start_time = time.time()
directory = "/content/drive/MyDrive/NNDL/dog cat/dog cat/train/cats"

for filename in os.listdir(directory):
    if filename.endswith(".jpg"):
        num_images += 1

        image = Image.open(os.path.join(directory, filename)).convert("L")
        image_array = np.array(image)

        fft_image = fftpack.fft2(image_array)
        fft_image_shifted = fftpack.fftshift(fft_image)

        rows, cols = image_array.shape
        center_row, center_col = rows // 2, cols // 2
        radius = 20
        high_pass_filter = np.ones((rows, cols))
        mask = np.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 = np.abs(fftpack.ifft2(fftpack.ifftshift(filtered_image_fft)))

total_time = time.time() - start_time

print("Number of images processed in sequence:", num_images)
print("Time taken for sequential processing:", total_time, "seconds")
```

```
Number of images processed in sequence: 279
Time taken for sequential processing: 28.283076286315918 seconds
```

Parallel

```
import os
import time
import cupy as cp
from PIL import Image


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: 279
Time taken for parallel processing: 4.5335447788238525 seconds
```