Lab Manual

of

High Performance Computing

(20DS509P)

By

Dev Jethva
23MDS003

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

Index

Sr. No.	Problem Statement	Date	Sign.
1.	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, and 20 and plot the speedup versus the number of threads. Compute the efficiency. • Display a visualization of performance comparison between serial, parallel and NumPY code. • Explain whether or not the scaling behavior is as expected.	17/1/2024	
2.	Write a program for Leibniz series for PI calculation to demonstrate the performance enhancement done by parallelizing the code through Open MP work-sharing of loops. Display a visualization of performance comparison between serial and parallel, a visual analysis of delay/speedup with the help of varying thread counts and maximum terms in the series for Pi value calculation. • Implement the code with different thread count and different maximum number of terms to be calculated for the series such as thread count 10, 20 and terms 100, 1000, 10000, 1000000. • Display a visualization of performance comparison between serial and parallel, a visual analysis of delay/speedup with the help of varying thread counts and maximum terms in the series for Pi value calculation.	24/1/2024	
3.	Implement Producer-Consumer problem (PCP). Analyze the significance of semaphore, mutex, bounded buffer, producer thread, and consumer thread using the code available on Producer-Consumer Problem in Python - AskPython. Demonstrate how PCP occurs for an application of your choice.	29/1/2024	
4.	Write a program to generate and print Fibonacci series, one thread must generate the series up to number and other thread must print them. Ensure proper synchronization.	31/1/2024	
5.	Consider a scenario where a person visits a supermarket for shopping. S/He purchases various items in different sections such as clothing, grocery, utensils. Write an OpenMP program to process the bill parallelly	31/1/2024	

	in each section and display the final amount to be paid by the customer. Analyze the time take by sequential and parallel processing.		
6.	Implement the following programs of OpenMPI • Print "Welcome to PDPU from process (processno_totalprocesses)". • Apply denoising algorithm to a set of n images with 4 processes. (n=4, 8). • Analyze time taken by serial and openMPI processes. • Try for 100 or more number of images.	7/2/2024	
7.	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.	14/2/2024	
8.	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.	14/2/2024	
9.	Calculate Pi value using openMPI send and receive messages for atleast 35-40 terms. Try the below mentioned commands, explain their task in one line and paste the output for each of them • Change the value on n as 2, 4, 8, 16. • Analyze the performance improvement using number of processes.	14/2/2024	
10.	Write a program to show collective communication by taking suitable example such that computing average of n numbers or computing sum or product of two matrices: • Bcast function • Scatter function • Gather function	14/2/2024	
11.	1. Describe Canon's Matrix Multiplication algorithm. 2. Implement Canon's Matrix Multiplication using collective communication. 3. Analyze the efficiency of the code.	19/2/2024	
12.	1. Write about derived data types used in MPI programming. 2. Steps to create and use derived data types. 3. Write its uses.	19/3/2024	

	4. Implement communication of derived data using one suitable example.			
13.	lshw (List Hardware) lsusb (List USB Devices) lspci (List PCI Devices) lsblk (List Block Devices) lscpu (List CPU) df (Disk Free) dmidecode (DMI Table Decode) ip a (IP Addresse)	top htop nvidia-smi lstopo perf numactl sar		
	For the given Python scripts that queries the CPU usage on a Linux-based system, understand the same and note the output for your device.		11/3/2024	
14.	Perform the following Image Processing Operations using the given images: • Image Blurring • Image Thresholding • Histogram based image analysis • Image Filtering/Denoising • Image Gray scaling		26/3/2024	
15.	Empirically understand and document the answers to the following: • What is CUDA? • What is the prerequisite for learning CUDA? • What are the languages that support CUDA? • What do you mean by a CUDA ready architecture? • How CUDA works? • What are the benefits and limitations of CUDA programming? • Understand and explain the CUDA program structure with an example. • Explain CUDA thread organization • Install and try CUDA sample program and explain the same. (installation steps)		1/4/2024	
16.	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 and 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		15/4/2024	
17.	Implement the following Image Processing operations in sequential and parallel using CUDA Programming. 1. Gaussian Blur • Describe Gaussian Blur in brief.		22/4/2024	

	• Where parallelism can be inserted? • Analyze the performance in serial and parallel model. 2. FFT- Fast Fourier Transform • Describe FFT in brief. • Where parallelism can be inserted? • Analyze the performance in serial and parallel model.		
18.	Final Learning Synopsis Submission		

Assignment 1

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

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

Using Numpy

```
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, out=result[start_idx:end_idx]))

        concurrent.futures.wait(futures)

    return result

matrix_sizes = [(5, 5), (50, 50), (100, 100), (250, 250),(500,500), (750, 750), (1000, 1000), (2000, 2000)]

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         0.006196       0.001795
1        (5, 5)        2         0.006196       0.000713
2        (5, 5)        4         0.006196       0.002331
3        (5, 5)        8         0.006196       0.005122
4        (5, 5)       10         0.006196       0.000956
5        (5, 5)       15         0.006196       0.001041
6        (5, 5)       20         0.006196       0.001321
7       (50, 50)       1         0.000064       0.000315
8       (50, 50)       2         0.000064       0.000433
9       (50, 50)       4         0.000064       0.000584
10      (50, 50)       8         0.000064       0.002852
11      (50, 50)      10         0.000064       0.003071
12      (50, 50)      15         0.000064       0.001474
13      (50, 50)      20         0.000064       0.001447
14    (100, 100)       1         0.003668       0.011495
15    (100, 100)       2         0.003668       0.002934
16    (100, 100)       4         0.003668       0.001123
17    (100, 100)       8         0.003668       0.001417
18    (100, 100)      10         0.003668       0.001704
19    (100, 100)      15         0.003668       0.003110
20    (100, 100)      20         0.003668       0.006952
21    (250, 250)       1         0.003142       0.007661
22    (250, 250)       2         0.003142       0.007903
23    (250, 250)       4         0.003142       0.007006
24    (250, 250)       8         0.003142       0.013687
25    (250, 250)      10         0.003142       0.013339
26    (250, 250)      15         0.003142       0.013645
27    (250, 250)      20         0.003142       0.058267
28    (500, 500)       1         0.030098       0.025425
29    (500, 500)       2         0.030098       0.041191
30    (500, 500)       4         0.030098       0.022950
31    (500, 500)       8         0.030098       0.027752
32    (500, 500)      10         0.030098       0.029450
33    (500, 500)      15         0.030098       0.021643
34    (500, 500)      20         0.030098       0.053042
35    (750, 750)       1         0.048432       0.029559
36    (750, 750)       2         0.048432       0.035149
37    (750, 750)       4         0.048432       0.036400
38    (750, 750)       8         0.048432       0.101134
39    (750, 750)      10         0.048432       0.074287
40    (750, 750)      15         0.048432       0.087460
41    (750, 750)      20         0.048432       0.110053
42  (1000, 1000)       1         0.099399       0.085901
43  (1000, 1000)       2         0.099399       0.090871
44  (1000, 1000)       4         0.099399       0.169034
45  (1000, 1000)       8         0.099399       0.178331
46  (1000, 1000)      10         0.099399       0.215278
47  (1000, 1000)      15         0.099399       0.249586
48  (1000, 1000)      20         0.099399       0.281935
49  (2000, 2000)       1         0.776949       0.784799
50  (2000, 2000)       2         0.776949       1.074981
51  (2000, 2000)       4         0.776949       0.966290
52  (2000, 2000)       8         0.776949       1.099694
53  (2000, 2000)      10         0.776949       1.024765
54  (2000, 2000)      15         0.776949       1.575939
55  (2000, 2000)      20         0.776949       1.325083
```

Using loop

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

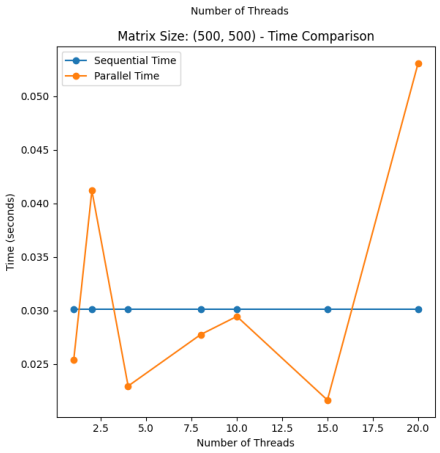
```
def multiply_matrix(A, B, result, start_row, end_row):
    try:
        for i in range(start_row, end_row):
            for j in range(N):
                result[i, j] = 0
                for k in range(N):
                    result[i, j] += A[i, k] * B[k, j]
    except NameError as e:
        pass


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)


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)


if __name__ == "__main__":
    main()


df2 = pd.read_csv('matrix_multiplication_results.csv')
df2
```

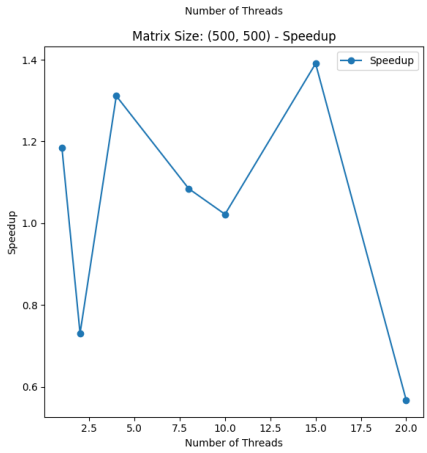
24	250	8	0.000118	0.000706	0.166554	0.020819
25	250	10	0.000118	0.000782	0.150259	0.015026
26	250	15	0.000118	0.001207	0.097373	0.006492
27	250	20	0.000118	0.001580	0.074415	0.003721
28	500	1	0.000134	0.000128	1.046729	1.046729
29	500	2	0.000134	0.000216	0.618102	0.309051
30	500	4	0.000134	0.000359	0.372340	0.093085
31	500	8	0.000134	0.000813	0.164319	0.020540
32	500	10	0.000134	0.000799	0.167164	0.016716
33	500	15	0.000134	0.001351	0.098800	0.006587
34	500	20	0.000134	0.001554	0.085916	0.004296
35	750	1	0.000125	0.000152	0.824176	0.824176
36	750	2	0.000125	0.000228	0.549738	0.274869
37	750	4	0.000125	0.000566	0.221239	0.055310
38	750	8	0.000125	0.000846	0.147929	0.018491
39	750	10	0.000125	0.001158	0.108114	0.010811
40	750	15	0.000125	0.001400	0.089392	0.005959
41	750	20	0.000125	0.001802	0.069444	0.003472
42	1000	1	0.000149	0.000146	1.022913	1.022913
43	1000	2	0.000149	0.000289	0.516102	0.258051
44	1000	4	0.000149	0.000408	0.365497	0.091374
45	1000	8	0.000149	0.000773	0.192723	0.024090
46	1000	10	0.000149	0.000959	0.155395	0.015540
47	1000	15	0.000149	0.001483	0.100466	0.006698
48	1000	20	0.000149	0.001828	0.081518	0.004076
49	2000	1	0.000226	0.000236	0.961538	0.961538
50	2000	2	0.000226	0.000352	0.643196	0.321598
51	2000	4	0.000226	0.000495	0.457611	0.114403
52	2000	8	0.000226	0.000849	0.266704	0.033338
53	2000	10	0.000226	0.000954	0.237322	0.023732
54	2000	15	0.000226	0.001438	0.157519	0.010501
55	2000	20	0.000226	0.001786	0.126853	0.006343

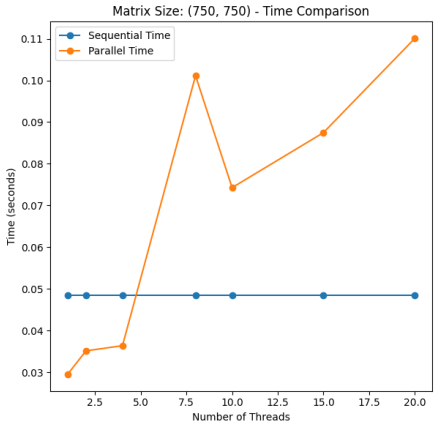
Display a visualization of performance comparison between serial, parallel and NumPY code

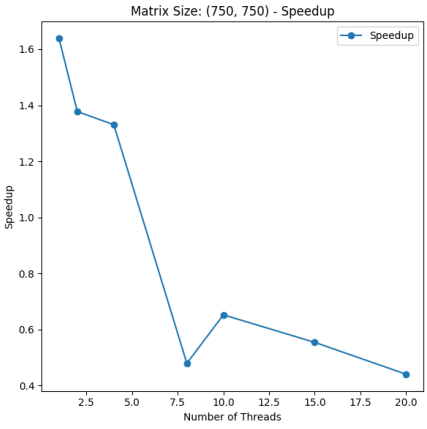
visualization of performance comparison between serial, parallel using NumPY code

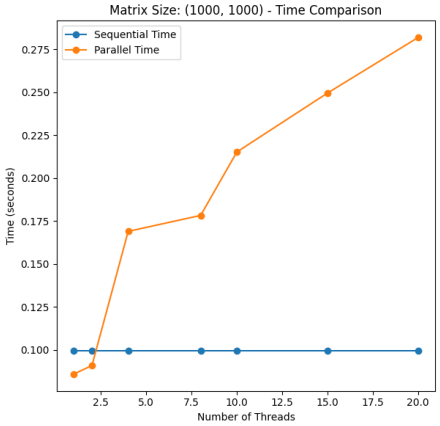
```
import matplotlib.pyplot as plt
matrix_sizes = df['Matrix Size'].unique()

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='Sequential Time')
    plt.plot(subset_df['Threads'], subset_df['Parallel Time'], marker='o', label='Parallel Time')
    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()
```

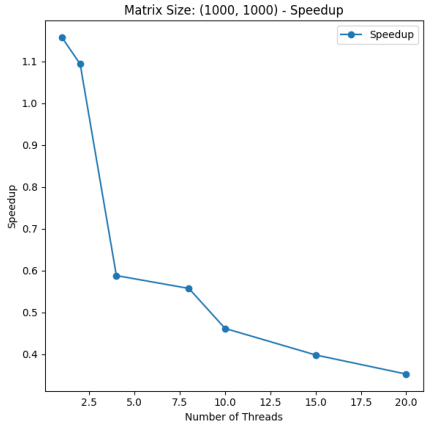












Matrix Size: (2000, 2000) - Time Comparison

Matrix Size: (2000, 2000) - Speedup

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.

In [2]:
```
import time
from multiprocessing import Pool

def calculate_pi_serial(num_iterations):
    pi = 0.0
    for i in range(num_iterations):
        term = 1.0 if i % 2 == 0 else -1.0
        pi += term / (2 * i + 1)
    return pi * 4

def calculate_pi_parallel_chunk(start, end):
    pi_chunk = 0.0
    for i in range(start, end):
        term = 1.0 if i % 2 == 0 else -1.0
        pi_chunk += term / (2 * i + 1)
    return pi_chunk

def calculate_pi_parallel(num_iterations, num_processes):
    chunk_size = num_iterations // num_processes
    pool = Pool(processes=num_processes)

    start_time = time.time()

    results = pool.starmap(calculate_pi_parallel_chunk, [(i * chunk_size, (i + 1) * c

    pi_parallel = sum(results) * 4

    end_time = time.time()

    print(f"Parallel PI: {pi_parallel}")
    print(f"Parallel Time: {end_time - start_time} seconds")

if __name__ == "__main__":
    NUM_ITERATIONS = 10000000
    NUM_PROCESSES = 4

    # Serial Calculation
    start_time = time.time()
    pi_serial = calculate_pi_serial(NUM_ITERATIONS)
    end_time = time.time()

    print(f"Serial PI: {pi_serial}")
    print(f"Serial Time: {end_time - start_time} seconds")

    # Parallel Calculation
    calculate_pi_parallel(NUM_ITERATIONS, NUM_PROCESSES)
```

```
Serial PI: 3.1415925535897915
Serial Time: 2.3656861782073975 seconds
Parallel PI: 3.1415925535897427
Parallel Time: 2.071720838546753 seconds
```

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.

In [3]:

```
import time
from multiprocessing import Pool

def calculate_pi_serial(num_iterations):
    pi = 0.0
    for i in range(num_iterations):
        term = 1.0 if i % 2 == 0 else -1.0
        pi += term / (2 * i + 1)
    return pi * 4

def calculate_pi_parallel_chunk(start, end):
    pi_chunk = 0.0
    for i in range(start, end):
        term = 1.0 if i % 2 == 0 else -1.0
        pi_chunk += term / (2 * i + 1)
    return pi_chunk

def calculate_pi_parallel(num_iterations, num_processes):
    chunk_size = num_iterations // num_processes
    pool = Pool(processes=num_processes)

    start_time = time.time()

    results = pool.starmap(calculate_pi_parallel_chunk, [(i * chunk_size, (i + 1) * c

    pi_parallel = sum(results) * 4

    end_time = time.time()

    print(f"Parallel PI with {num_processes} threads and {num_iterations} terms: {pi_
    print(f"Parallel Time: {end_time - start_time} seconds")

if __name__ == "__main__":
    thread_counts = [10, 20]
    term_counts = [100, 1000, 10000, 1000000]

    for threads in thread_counts:
        for terms in term_counts:
            print(f"\nThread Count: {threads}, Max Terms: {terms}")

            # Serial Calculation
            start_time = time.time()
            pi_serial = calculate_pi_serial(terms)
            end_time = time.time()

            print(f"Serial PI: {pi_serial}")
            print(f"Serial Time: {end_time - start_time} seconds")
```

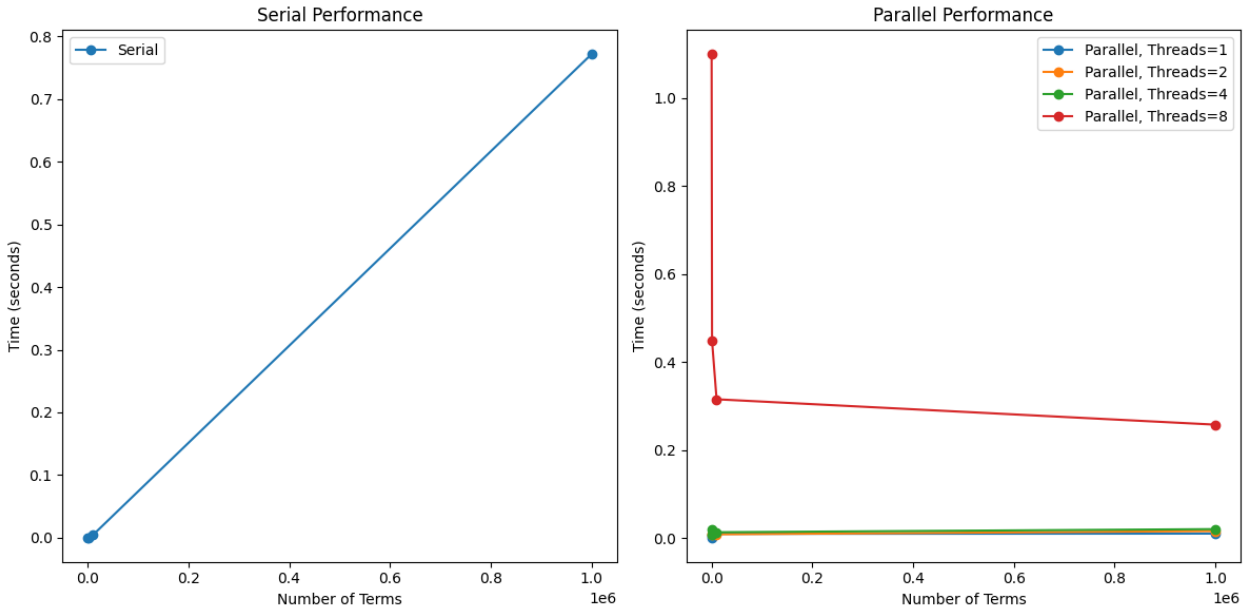
```
            # Parallel Calculation
            calculate_pi_parallel(terms, threads)
```

```
Thread Count: 10, Max Terms: 100
Serial PI: 3.1315929035585537
Serial Time: 3.0040740966796875e-05 seconds
Parallel PI with 10 threads and 100 terms: 3.131592903558554
Parallel Time: 0.009278535842895508 seconds

Thread Count: 10, Max Terms: 1000
Serial PI: 3.140592653839794
Serial Time: 0.00036525726318359375 seconds
Parallel PI with 10 threads and 1000 terms: 3.1405926538397937
Parallel Time: 0.007089138031005859 seconds

Thread Count: 10, Max Terms: 10000
Serial PI: 3.1414926535900345
Serial Time: 0.0038022994995117188 seconds
Parallel PI with 10 threads and 10000 terms: 3.1414926535900447
Parallel Time: 0.015659332275390625 seconds

Thread Count: 10, Max Terms: 1000000
Serial PI: 3.1415916535897743
Serial Time: 0.20785188674926758 seconds
Parallel PI with 10 threads and 1000000 terms: 3.1415916535897197
Parallel Time: 0.22098231315612793 seconds

Thread Count: 20, Max Terms: 100
Serial PI: 3.1315929035585537
Serial Time: 3.361701965332031e-05 seconds
Parallel PI with 20 threads and 100 terms: 3.131592903558554
Parallel Time: 0.010380983352661133 seconds

Thread Count: 20, Max Terms: 1000
Serial PI: 3.140592653839794
Serial Time: 0.00032448768615722656 seconds
Parallel PI with 20 threads and 1000 terms: 3.1405926538397932
Parallel Time: 0.0035829544067382812 seconds

Thread Count: 20, Max Terms: 10000
Serial PI: 3.1414926535900345
Serial Time: 0.00205230712890625 seconds
Parallel PI with 20 threads and 10000 terms: 3.1414926535900434
Parallel Time: 0.007832765579223633 seconds

Thread Count: 20, Max Terms: 1000000
Serial PI: 3.1415916535897743
Serial Time: 0.1933746337890625 seconds
Parallel PI with 20 threads and 1000000 terms: 3.14159165358978
Parallel Time: 0.22645282745361328 seconds
```

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.

In [4]:
```
import time
import matplotlib.pyplot as plt
from multiprocessing import Pool

def calculate_pi_serial(num_iterations):
    pi = 0.0
    for i in range(num_iterations):
        term = 1.0 if i % 2 == 0 else -1.0
        pi += term / (2 * i + 1)
    return pi * 4

def calculate_pi_parallel_chunk(start, end):
    pi_chunk = 0.0
    for i in range(start, end):
        term = 1.0 if i % 2 == 0 else -1.0
        pi_chunk += term / (2 * i + 1)
    return pi_chunk

def calculate_pi_parallel(num_iterations, num_processes):
    chunk_size = num_iterations // num_processes
    pool = Pool(processes=num_processes)

    start_time = time.time()

    results = pool.starmap(calculate_pi_parallel_chunk, [(i * chunk_size, (i + 1) * c

    pi_parallel = sum(results) * 4

    end_time = time.time()

    return pi_parallel, end_time - start_time

def plot_performance(thread_counts, term_counts):
    serial_times = []
    parallel_times = []

    for terms in term_counts:
        # Serial Calculation
        start_time = time.time()
        calculate_pi_serial(terms)
        end_time = time.time()
        serial_times.append(end_time - start_time)

        # Parallel Calculation
        for threads in thread_counts:
            _, parallel_time = calculate_pi_parallel(terms, threads)
            parallel_times.append(parallel_time)

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

    # Plot Serial Time
    plt.subplot(1, 2, 1)
    plt.plot(term_counts, serial_times, marker='o', label='Serial')
    plt.title('Serial Performance')
    plt.xlabel('Number of Terms')
```

```
    plt.ylabel('Time (seconds)')
    plt.legend()

    # Plot Parallel Time
    plt.subplot(1, 2, 2)
    for i, threads in enumerate(thread_counts):
        plt.plot(term_counts, parallel_times[i * len(term_counts):(i + 1) * len(term_
    plt.title('Parallel Performance')
    plt.xlabel('Number of Terms')
    plt.ylabel('Time (seconds)')
    plt.legend()

    plt.tight_layout()
    plt.show()

if __name__ == "__main__":
    thread_counts = [1, 2, 4, 8]
    term_counts = [100, 1000, 10000, 1000000]

    plot_performance(thread_counts, term_counts)
```



In [ ]:

1 Assignment 3

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

Ans >

The producer-consumer problem is a 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:

- Shared Buffer:
  - A fixed-size buffer is utilized, acting as a common storage space for both producers and consumers.
- 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.
- 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.

- Illustration with a Different 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.

- Shared Buffer (Kitchen Counter):
  - Represents the kitchen counter where the prepared dishes are temporarily stored.
- 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.
- 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.

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

```

In [2]:
```
import threading
import time

```

In [3]:
```
# Shared Memory variables
CAPACITY = 10
buffer = [-1 for i in range(CAPACITY)]
in_index = 0
out_index = 0
```

In [4]:
```
# Declaring Semaphores
mutex = threading.Semaphore()
empty = threading.Semaphore(CAPACITY)
full = threading.Semaphore(0)
```

In [5]:
```
# 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

```

In [6]:
```
# 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
```

In [7]:
```
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
```

In [8]:
```
import time
```

In [9]:
```
CAPACITY = 10
buffer = [-1 for i in range(CAPACITY)]
in_index = 0
out_index = 0
```

In [10]:
```
mutex = threading.Semaphore()
empty = threading.Semaphore(CAPACITY)
full = threading.Semaphore(0)

```

In [11]:
```
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)

```

In [12]:
```
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)
```

In [13]:
```
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
```

In [14]:
```
import queue
import random
```

In [15]:
```
MAX_QUEUE_SIZE = 5
event_queue = queue.Queue(MAX_QUEUE_SIZE)
mutex = threading.Lock()
empty = threading.Semaphore(MAX_QUEUE_SIZE)
full = threading.Semaphore(0)
```

In [16]:
```
class UserClickProducer(threading.Thread):
 def run(self):
  global MAX_QUEUE_SIZE, event_queue
  global mutex, empty, full
  for _ in range(10):
   print("User clicked")
   empty.acquire()
   mutex.acquire()
   event_queue.put("Click")
   mutex.release()
   full.release()
   time.sleep(random.uniform(0.1, 0.5))

```

In [17]:
```
class EventHandlerConsumer(threading.Thread):
 def run(self):
  global MAX_QUEUE_SIZE, event_queue
  global mutex, empty, full
  for _ in range(10):
   full.acquire()
   mutex.acquire()
   event = event_queue.get()
   print(f"Handling event: {event}")
   mutex.release()
   empty.release()
   time.sleep(random.uniform(0.1, 0.5))

```

In [18]:
```
user_click_producer = UserClickProducer()
event_handler_consumer = EventHandlerConsumer()
user_click_producer.start()
event_handler_consumer.start()
user_click_producer.join()
event_handler_consumer.join()

```

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

In [19]:
```
import queue
import random
```

Example

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

In [20]:
```
MAX_QUEUE_SIZE = 5
event_queue = queue.Queue(MAX_QUEUE_SIZE)
mutex = threading.Lock()
empty = threading.Semaphore(MAX_QUEUE_SIZE)
full = threading.Semaphore(0)
```

In [21]:
```
class UserClickProducer(threading.Thread):
    def run(self):
        global MAX_QUEUE_SIZE, event_queue
        global mutex, empty, full
        for _ in range(10):
            print("User clicked")
empty.acquire()
mutex.acquire()
event_queue.put("Click")
mutex.release()
full.release()
time.sleep(random.uniform(0.1, 0.5))
```

In [22]:
```
class EventHandlerConsumer(threading.Thread):
    def run(self):
        global MAX_QUEUE_SIZE, event_queue
        global mutex, empty, full
        for _ in range(10):
            full.acquire()
            mutex.acquire()
            event = event_queue.get()
            print(f"Handling event: {event}")
            mutex.release()
            empty.release()
            time.sleep(random.uniform(0.1, 0.5))
```

In [ ]:
```
user_click_producer = UserClickProducer()
event_handler_consumer = EventHandlerConsumer()
user_click_producer.start()
event_handler_consumer.start()
user_click_producer.join()
event_handler_consumer.join()
```

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

In [ ]:
```

```

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.

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

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

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

In [4]:
```
def main():
    n = int(input("Enter the number of Fibonacci numbers to generate: "))

    fib_list = []
    lock = threading.Lock()
    start_time = time.time()

    # Create two threads
    generate_thread = threading.Thread(target=generate_fibonacci, args=(n, fib_list,
    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: 20
0, Thread ID 1880 at 2024-02-01 16:18:35: , Time Taken: 0.000721 seconds
1, Thread ID 1880 at 2024-02-01 16:18:35: , Time Taken: 0.000721 seconds
1, Thread ID 1880 at 2024-02-01 16:18:35: , Time Taken: 0.000721 seconds
2, Thread ID 1880 at 2024-02-01 16:18:35: , Time Taken: 0.000721 seconds
3, Thread ID 1880 at 2024-02-01 16:18:35: , Time Taken: 0.000721 seconds
5, Thread ID 1880 at 2024-02-01 16:18:35: , Time Taken: 0.000721 seconds
8, Thread ID 1880 at 2024-02-01 16:18:35: , Time Taken: 0.000721 seconds
13, Thread ID 1880 at 2024-02-01 16:18:35: , Time Taken: 0.000721 seconds
21, Thread ID 1880 at 2024-02-01 16:18:35: , Time Taken: 0.000721 seconds
34, Thread ID 1880 at 2024-02-01 16:18:35: , Time Taken: 0.000721 seconds
55, Thread ID 1880 at 2024-02-01 16:18:35: , Time Taken: 0.000721 seconds
89, Thread ID 1880 at 2024-02-01 16:18:35: , Time Taken: 0.000721 seconds
144, Thread ID 1880 at 2024-02-01 16:18:35: , Time Taken: 0.000721 seconds
233, Thread ID 1880 at 2024-02-01 16:18:35: , Time Taken: 0.000721 seconds
377, Thread ID 1880 at 2024-02-01 16:18:35: , Time Taken: 0.000721 seconds
610, Thread ID 1880 at 2024-02-01 16:18:35: , Time Taken: 0.000721 seconds
987, Thread ID 1880 at 2024-02-01 16:18:35: , Time Taken: 0.000721 seconds
1597, Thread ID 1880 at 2024-02-01 16:18:35: , Time Taken: 0.000721 seconds
2584, Thread ID 1880 at 2024-02-01 16:18:35: , Time Taken: 0.000721 seconds
4181, Thread ID 1880 at 2024-02-01 16:18:35: , Time Taken: 0.000721 seconds
```

In [ ]:

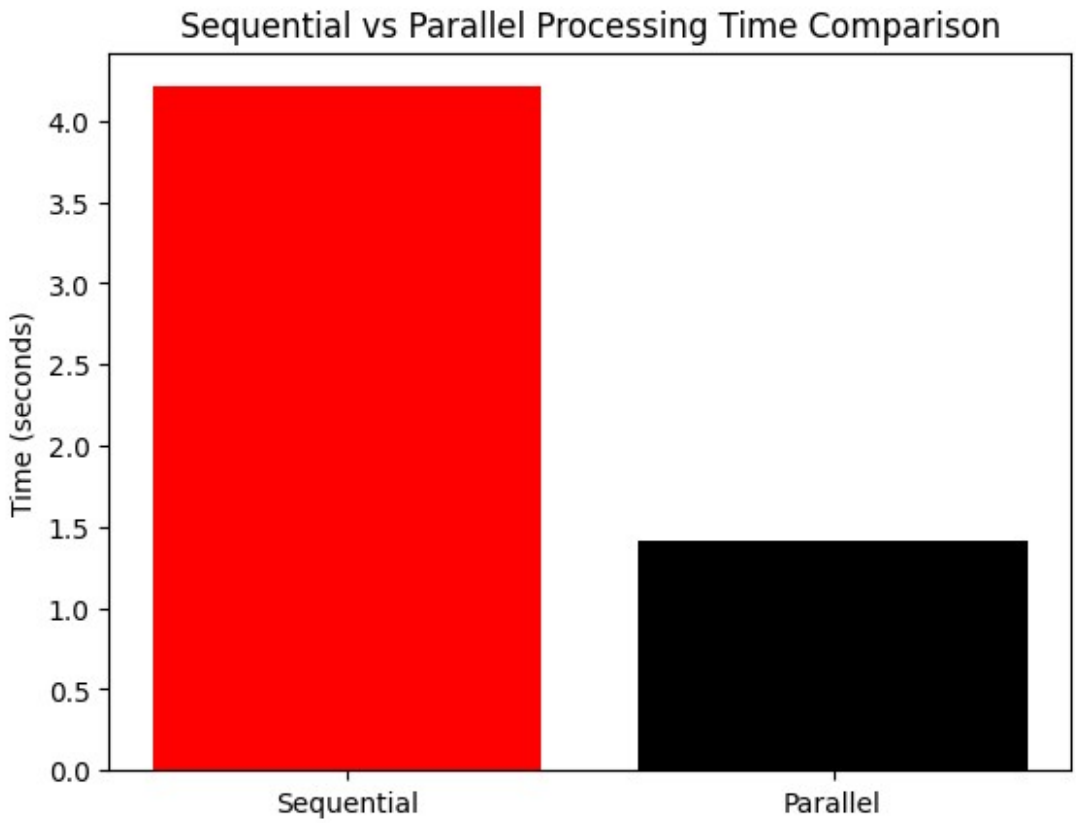
Assignment 5

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.

```
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

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 22  # 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 10  # 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 5  # Cost of each utensils item

if __name__ == "__main__":
    # Sequential Processing
    start_time = time.time()
```

```
    clothing_cost = sum(process_clothing() for _ in range(7))
    grocery_cost = sum(process_grocery() for _ in range(7))
    utensils_cost = sum(process_utensils() for _ in range(7))

    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 = 7
    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=['Red', 'Black'])
    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 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...
Total amount to be paid (Sequential): $259.00
Time taken (Sequential): 4.21 seconds

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

Processing utensils item...
Processing grocery item...
Processing clothing item...
Processing utensils item...
Processing grocery item...
Processing clothing item...
Processing utensils item...
Processing grocery item...
Processing clothing item...
Processing utensils item...
Processing grocery item...
Processing clothing item...
Processing utensils item...
Total amount to be paid (Parallel): $259.00
Time taken (Parallel): 1.41 seconds
```



According to the graph, parallel processing takes less time than sequential processing since each process runs on a separate thread, but sequential processing takes nearly three times as long.

HPC Assignment 6

February 8, 2024

1 HPC_LabAssignment6(openMPI)

1.0.1 a)Print "Welcome to PDPU from process (processno_totalprocesses)".

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

1.0.3 c)Analyze time taken by serial and openMPI processes.

1.0.4 d)Try for 100 or more number of images.

[38]:
```
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
```

[19]:
```
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
```

[20]:
```
!mpiexec -n 4 python mpi_script.py
```

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

[22]:
```
!mpiexec -n 8 python mpi_script.py
```

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

[32]:
```
def add_noise(image):
    noisy_image = image + np.random.normal(loc=0, scale=10, size=image.shape)
    return np.clip(noisy_image, 0, 255).astype(np.uint8)

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

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

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

    folder_path = "D:/code/HPC/Assingment/Assignment6/image/formula_1_racing"
    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: 100
Serial Denoising:
Time taken for serial denoising: 37.53 seconds
Parallel Denoising:
Time taken for parallel denoising: 37.22 seconds
```

[35]:
```
!mpiexec -n 4 python 100_images.py
```

```
Number of images: 100
Serial Denoising:
Time taken for serial denoising: 48.61 seconds
Parallel Denoising:
Time taken for parallel denoising: 12.04 seconds
```

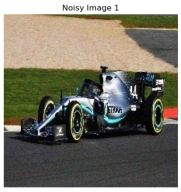
[36]:
```
!mpiexec -n 8 python 100_images.py
```

```
Number of images: 100
Serial Denoising:
```

```
Time taken for serial denoising: 60.98 seconds
Parallel Denoising:
Time taken for parallel denoising: 7.57 seconds
```

[43]:
```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 = "D:/code/HPC/Assingment/Assignment6/image/formula_1_racing/
 ↪output1/noisy"
denoised_folder = "D:/code/HPC/Assingment/Assignment6/image/formula_1_racing/
 ↪output1/denoised"
plot_images_side_by_side(noisy_folder, denoised_folder)
```

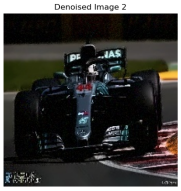
Noisy Image 1



Denoised Image 1

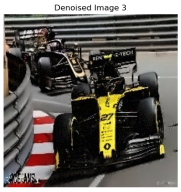
Noisy Image 2

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

HPC-12-2

February 14, 2024

1 Assignment 7

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

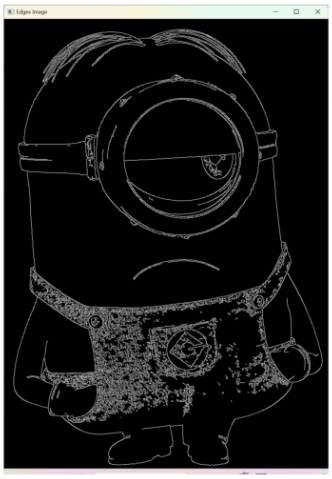
```
rank = 3 , Displayed the grayscale image
rank = 3 , Displayed the edges image
```

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

[19]:
```python
# Load the images
image1 = cv2.imread("hpc-7-2-2.png")
image2 = cv2.imread("hpc-7-2-3.png")
image3 = cv2.imread("hpc-7-2-1.png")

# Convert BGR to RGB (Matplotlib uses RGB)
image1_rgb = cv2.cvtColor(image1, cv2.COLOR_BGR2RGB)
image2_rgb = cv2.cvtColor(image2, cv2.COLOR_BGR2RGB)
image3_rgb = cv2.cvtColor(image3, cv2.COLOR_BGR2RGB)

# Display the images
plt.figure(figsize=(15, 10))

plt.subplot(1, 3, 1)
plt.imshow(image1_rgb)
plt.axis('off')
plt.title('grayscaled image')

plt.subplot(1, 3, 2)
plt.imshow(image2_rgb)
plt.axis('off')
plt.title('edges of image')

plt.subplot(1, 3, 3)
plt.imshow(image3_rgb)
plt.axis('off')
plt.title('original image')

plt.show()
```

grayscaled image

edges of image



original image

HPC-14-2

February 14, 2024

1 Assignment 8

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

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

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

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

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

[ ]:
```
if rank == 0:
    message = "Hello from process 0"
    comm.send(message, dest=1)

    received_message = comm.recv(source=1)
    print(f"Process 0 received message: {received_message}")

elif rank == 1:
    received_message = comm.recv(source=0)
    print(f"Process 1 received message: {received_message}")

    reply = "Hello from process 1"
    comm.send(reply, dest=0)
```

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

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

[ ]:
```
if rank == 0:
    message = "Hello from process 0 (Async)"
    req_send = comm.isend(message, dest=1)  # 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}")
```

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

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

Assignment 9

February 16, 2024

1 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.

[3]:
```
from mpi4py import MPI
import random
import matplotlib.pyplot as plt
import time
```

[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 = 16.853801012039185 seconds
Number of processes: 1
Estimated Pi: 3.141592643589326
Execution time: 16.853801012039185 seconds
```

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

```
Process 1 calculated: Pi = -18.813394448175345, Time = 7.08020806312561 seconds
Process 0 calculated: Pi = 21.954987091759833, Time = 7.084234952926636 seconds
Process 1 received: Pi = -18.813394448175345, Time = 7.08020806312561 seconds
Number of processes: 2
Estimated Pi: 3.141592643584488
Execution time: 7.084234952926636 seconds
```

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

```
Process 1 calculated: Pi = -9.894192713487952, Time = 5.464809417724609 seconds
Process 2 calculated: Pi = 9.243547576205538, Time = 5.615514755249023 seconds
Process 3 calculated: Pi = -8.919201734688878, Time = 5.6487884521484375 seconds
Process 0 calculated: Pi = 12.711439515567903, Time = 5.480837345123291 seconds
Process 1 received: Pi = -9.894192713487952, Time = 5.4658119678497314 seconds
Process 2 received: Pi = 9.243547576205538, Time = 5.615514755249023 seconds
Process 3 received: Pi = -8.919201734688878, Time = 5.6487884521484375 seconds
Number of processes: 4
Estimated Pi: 3.1415926435966117
Execution time: 5.6487884521484375 seconds
```

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

```
Process 5 calculated: Pi = -4.399299923327154, Time = 4.300693988800049 seconds
Process 4 calculated: Pi = 4.5064163512322155, Time = 4.332261800765991 seconds
Process 3 calculated: Pi = -4.662641756747161, Time = 4.349788427352905 seconds
Process 2 calculated: Pi = 4.924086190869872, Time = 4.361311197280884 seconds
Process 6 calculated: Pi = 4.319461385334904, Time = 4.376924514770508 seconds
Process 1 calculated: Pi = -5.494892790161891, Time = 4.4154744148254395 seconds
Process 7 calculated: Pi = -4.256559977941305, Time = 4.42398738861084 seconds
Process 0 calculated: Pi = 8.205023164331104, Time = 4.431013584136963 seconds
Process 1 received: Pi = -5.494892790161891, Time = 4.4154744148254395 seconds
Process 2 received: Pi = 4.924086190869872, Time = 4.361311197280884 seconds
Process 3 received: Pi = -4.662641756747161, Time = 4.349788427352905 seconds
```

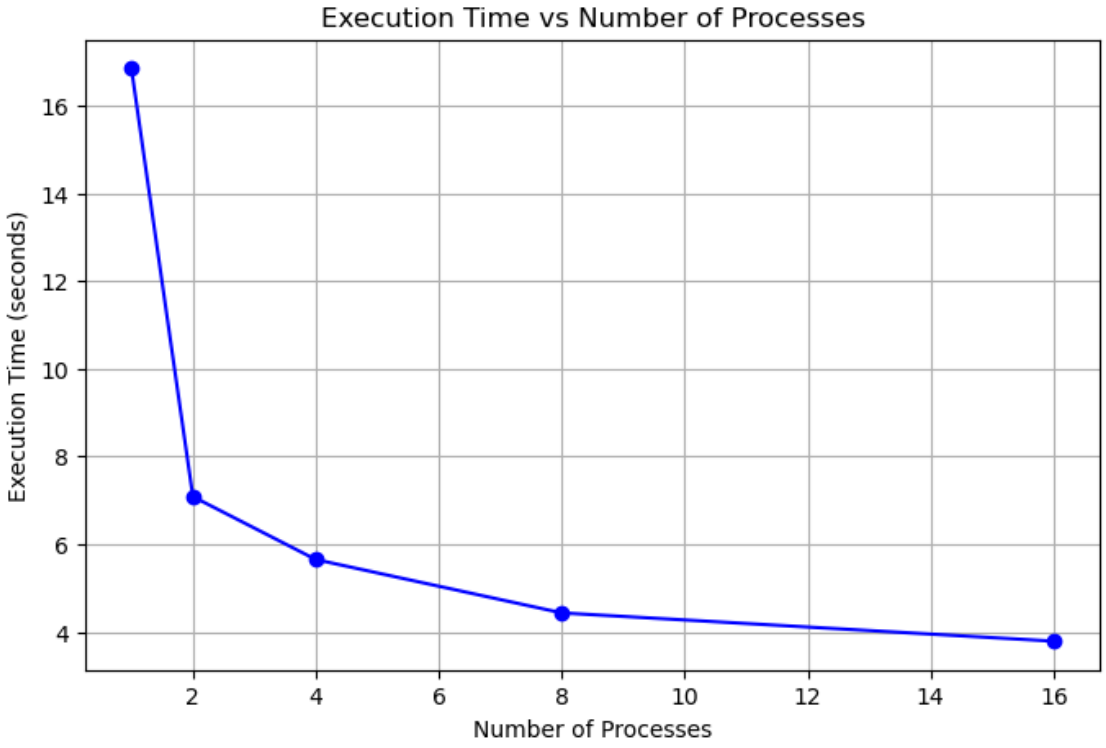
```
Process 4 received: Pi = 4.5064163512322155, Time = 4.332261800765991 seconds
Process 5 received: Pi = -4.399299923327154, Time = 4.301691293716431 seconds
Process 6 received: Pi = 4.319461385334904, Time = 4.376924514770508 seconds
Process 7 received: Pi = -4.256559977941305, Time = 4.42398738861084 seconds
Number of processes: 8
Estimated Pi: 3.1415926435905845
Execution time: 4.432020425796509 seconds
```

[9]:
```
!mpiexec -n 16 python Calculate_Pi_value.py
```

```
Process 15 calculated: Pi = -2.0347399856201327, Time = 2.499709367752075
seconds
Process 14 calculated: Pi = 2.048883162920837, Time = 2.5267527103424072 seconds
Process 6 calculated: Pi = 2.2705782224129933, Time = 2.5746617317199707 seconds
Process 11 calculated: Pi = -2.102010079259159, Time = 2.591703176498413 seconds
Process 7 calculated: Pi = -2.221819992321798, Time = 2.648799180984497 seconds
Process 2 calculated: Pi = 2.799255131066239, Time = 2.6804044246673584 seconds
Process 3 calculated: Pi = -2.5606316774894724, Time = 2.7119908332824707
seconds
Process 13 calculated: Pi = -2.064566837527382, Time = 2.957490921020508 seconds
Process 5 calculated: Pi = -2.334733085800057, Time = 3.46201753616333 seconds
Process 10 calculated: Pi = 2.1248310598041242, Time = 3.718217134475708 seconds
Process 1 calculated: Pi = -3.343438528087643, Time = 3.7381417751312256 seconds
Process 9 calculated: Pi = -2.1514542620738304, Time = 3.7782135009765625
seconds
Process 8 calculated: Pi = 2.1831425099178574, Time = 3.7782235145568848 seconds
Process 4 calculated: Pi = 2.4242898780367725, Time = 3.769716739654541 seconds
Process 12 calculated: Pi = 2.0821264731939486, Time = 3.782113790512085 seconds
Process 0 calculated: Pi = 6.021880654414, Time = 3.7436184883117676 seconds
Process 1 received: Pi = -3.343438528087643, Time = 3.7381417751312256 seconds
Process 2 received: Pi = 2.799255131066239, Time = 2.6804044246673584 seconds
Process 3 received: Pi = -2.5606316774894724, Time = 2.7119908332824707 seconds
Process 4 received: Pi = 2.4242898780367725, Time = 3.769716739654541 seconds
Process 5 received: Pi = -2.334733085800057, Time = 3.46201753616333 seconds
Process 6 received: Pi = 2.2705782224129933, Time = 2.5746617317199707 seconds
Process 7 received: Pi = -2.221819992321798, Time = 2.648799180984497 seconds
Process 8 received: Pi = 2.1831425099178574, Time = 3.7782235145568848 seconds
Process 9 received: Pi = -2.1514542620738304, Time = 3.7782135009765625 seconds
Process 10 received: Pi = 2.1248310598041242, Time = 3.718217134475708 seconds
Process 11 received: Pi = -2.102010079259159, Time = 2.591703176498413 seconds
Process 12 received: Pi = 2.0821264731939486, Time = 3.782113790512085 seconds
Process 13 received: Pi = -2.064566837527382, Time = 2.957490921020508 seconds
Process 14 received: Pi = 2.048883162920837, Time = 2.5267527103424072 seconds
Process 15 received: Pi = -2.0347399856201327, Time = 2.499709367752075 seconds
Number of processes: 16
Estimated Pi: 3.141592643587298
Execution time: 3.7876338958740234 seconds
```

[4]:
```
num_processes = [1,2, 4, 8, 16]
execution_times = [ 16.853801012039185, 7.084234952926636, 5.6487884521484375,␣
↪4.432020425796509, 3.7876338958740234]
```

[5]:
```
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()
```



Assignment 10

February 16, 2024

1 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

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

[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: 592
```

[9]:
```
!mpiexec -n 4 python Bcast.py
```

```
Root process (Rank 0) is broadcasting data to other processes…
Process 0 received data: 29
Process 1 is waiting to receive broadcasted data from the root process (Rank 0)
Process 1 received data: 29
Process 2 is waiting to receive broadcasted data from the root process (Rank 0)
Process 2 received data: 29
Process 3 is waiting to receive broadcasted data from the root process (Rank 0)
Process 3 received data: 29
```

[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 (Rank 0)")

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

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

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

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

[8]:
```
!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
```

[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: [38]
```

[10]:
```
!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: [85 87  2 52]
```

Assignment 11

February 21, 2024

1 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.

[8]:
```
from mpi4py import MPI
import numpy as np
import time
import matplotlib.pyplot as plt
```

[2]:
```
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_multiplication, "seconds")
```

```
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:
[[39301 38115 39276 … 39972 40525 40039]
 [40443 39184 39780 … 40613 40866 40722]
 [40396 39733 39094 … 41001 40674 40047]
 …
 [40679 39558 40017 … 40813 41857 40597]
```

```
 [41755 38983 39306 … 40994 41444 40810]
 [39028 37840 38002 … 39761 40392 39509]]
Broadcasting time: 0.0170133113861084 seconds
Matrix multiplication time: 36.74707531929016 seconds
```

[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:
[[40890 41533 40346 … 41776 41360 40546]
 [39549 39305 39292 … 40792 39918 38704]
 [39597 39197 39112 … 40250 39542 39111]
 …
 [40889 40694 39536 … 41066 40201 39835]
 [41301 40419 39929 … 41947 41219 40105]
 [39774 39226 39003 … 40789 39911 38739]]
Broadcasting time: 0.04213356971740723 seconds
Matrix multiplication time: 12.16959547996521 seconds
```

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

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

```
Rank 6: Starting execution
Rank 6: Broadcasting matrices A and B
Rank 6: Matrices A and B broadcasted
Rank 4: Starting execution
Rank 4: Broadcasting matrices A and B
Rank 4: Matrices A and B broadcasted
Rank 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:
[[39367 40313 39846 … 39423 39870 39227]
 [39930 40975 39596 … 39867 40595 39561]
 [38730 40191 39867 … 39195 40115 39511]
 …
 [38345 39574 38339 … 39128 39093 38667]
 [40160 40961 39564 … 40825 41197 40526]
 [39411 39998 40239 … 40304 40227 39382]]
Broadcasting time: 0.0722498893737793 seconds
Matrix multiplication time: 9.310021162033081 seconds
```

[5]:
```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_multiplication, "seconds")
```

```
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:
[[39286 40520 41025 … 39506 40080 40731]
 [39804 40085 41133 … 40010 40795 41125]
 [39301 40436 39992 … 39477 39718 39905]
 …
 [39685 42441 41165 … 40145 41046 41222]
 [39311 40625 41750 … 40231 40082 40267]
 [40596 41792 41829 … 40820 41994 41560]]
Broadcasting time: 0.016011714935302734 seconds
Gathering time: 0.002999544143676758 seconds
Matrix multiplication time: 37.09537482261658 seconds
```

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

```
Rank 3: Starting execution
```

```
Rank 3: Broadcasting matrices A and B
Rank 3: Matrices A and B broadcasted
Rank 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:
[[40390 41613 42308 … 39966 40617 40247]
 [39805 39909 41212 … 40205 40558 41291]
 [41095 40848 41587 … 39871 40262 40793]
 …
 [41147 41114 41982 … 40127 40672 40632]
 [40029 39824 40773 … 40097 39583 40377]
 [39479 40690 40869 … 39596 40096 39963]]
Broadcasting time: 0.037689924240112305 seconds
Gathering time: 0.06966972351074219 seconds
Matrix multiplication time: 12.694447040557861 seconds
Rank 2: Starting execution
Rank 2: Broadcasting matrices A and B
Rank 2: Matrices A and B broadcasted
Rank 1: Starting execution
Rank 1: Broadcasting matrices A and B
Rank 1: Matrices A and B broadcasted
```

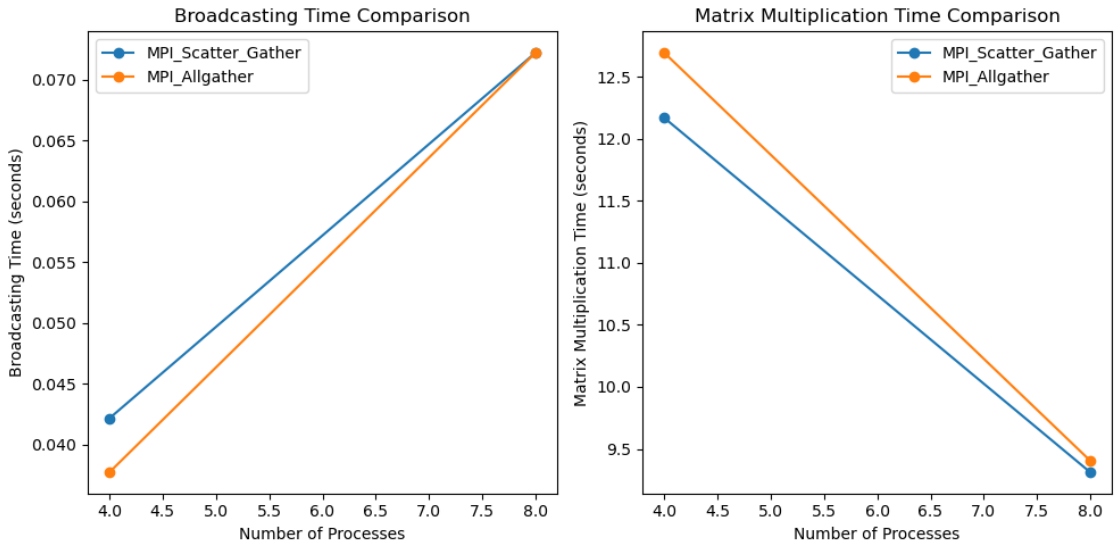
[7]:
```
!mpiexec -n 8 python MPI_Allgather.py
```

```
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 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 7: Starting execution
Rank 7: Broadcasting matrices A and B
Rank 7: 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:
```

```
[[39256 40549 38907 … 40831 40266 40966]
 [40495 39807 40080 … 41006 40655 41473]
 [39952 40595 39476 … 40758 39646 40756]
 …
 [39402 39458 39381 … 39838 40326 40670]
 [39833 39651 39985 … 40758 40584 41711]
 [40211 39729 39469 … 40826 40539 40597]]
Broadcasting time: 0.0722506046295166 seconds
Gathering time: 0.3540804386138916 seconds
Matrix multiplication time: 9.402881145477295 seconds
Rank 4: Starting execution
Rank 4: Broadcasting matrices A and B
Rank 4: Matrices A and B broadcasted
Rank 3: Starting execution
Rank 3: Broadcasting matrices A and B
Rank 3: Matrices A and B broadcasted
```

[11]:
```
scatter_gather_processes = [4, 8]
scatter_gather_broadcasting_time = [0.04213356971740723, 0.0722498893737793]
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_Scatter_Gather')
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='MPI_Scatter_Gather')
plt.plot(allgather_processes, allgather_multiplication_time, marker='o',␣
 ↪label='MPI_Allgather')
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.

HPC 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.

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(42, 99)

# 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()

!mpiexec -n 4 python 12.py

Process 0 received data: x=42, y=99
Process 2 received data: x=42, y=99
Process 1 received data: x=42, y=99
Process 3 received data: x=42, y=99
```

HPC ASSIGNMENT 13

lshw:

Display detailed hardware component information.

Input: - sudo lshw

Output: -

[sudo] password for user:

laptop-6in5e1d3

description: Computer

width: 64 bits

capabilities: smp vsyscall32

*-core

description: Motherboard

physical id: 0

*-memory

description: System memory

physical id: 0

size: 8GiB

*-cpu

product: Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz

vendor: Intel Corp.

physical id: 1

bus info: cpu@0

version: 6.165.2

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 x2apic 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

*-display:0

description: 3D controller

product: Microsoft Corporation

vendor: Microsoft Corporation

physical id: 2

bus info: pci@6274:00:00.0

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: 3

bus info: pci@673b: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-dfffffff

*-virtio1 UNCLAIMED

description: Virtual I/O device

physical id: 0

bus info: virtio@1

configuration: driver=virtiofs

*-display:1

description: 3D controller

product: Microsoft Corporation

vendor: Microsoft Corporation

physical id: 4

bus info: pci@971b:00:00.0

version: 00

width: 32 bits

clock: 33MHz

capabilities: bus_master cap_list

configuration: driver=dxgkrnl latency=0

resources: irq:0

*-scsi:0

description: SCSI storage controller

product: Virtio console

```
        vendor: Red Hat, Inc.
        physical id: 5
        bus info: pci@9def: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
   *-pnp00:00
        product: PnP device PNP0b00
        physical id: 6
        capabilities: pnp
        configuration: driver=rtc_cmos
   *-scsi:1
        physical id: 7
        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: f9b3fa53-afbd-4af6-99b9-b7ce3182ac64

size: 2GiB

capacity: 2GiB

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: dca79db0-6b6e-4726-8879-ad73e977ae75

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 09:26:36 filesystem=ext4 lastmountpoint=/distro logicalsectorsize=512 modified=2024-03-13 21:03:01 mount.fstype=ext4 mount.options=rw,relatime,discard,errors=remount-ro,data=ordered mounted=2024-03-13 21:03:01 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:25:86:ef

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=172.27.192.178 link=yes multicast=yes speed=10Gbit/s

lsusb:

List connected USB devices.

Input: -lsusb

Output: -

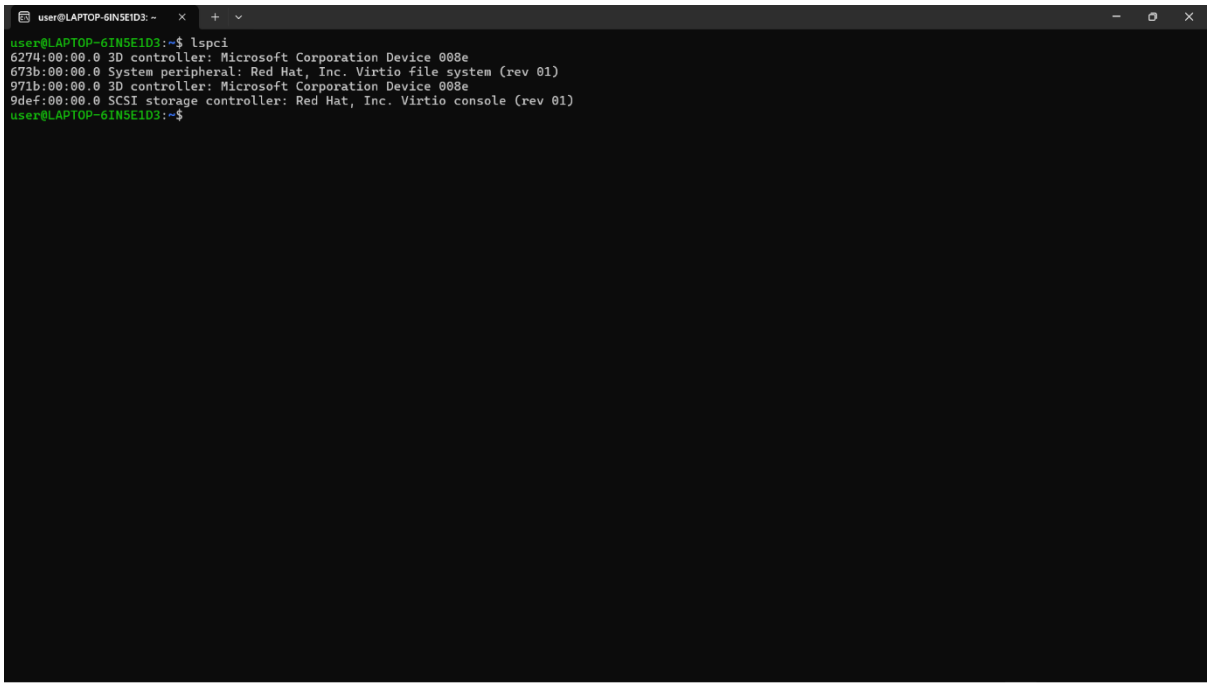
```
user@LAPTOP-6IN5E1D3:~$ lsusb
Bus 002 Device 001: ID 1d6b:0003 Linux Foundation 3.0 root hub
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
user@LAPTOP-6IN5E1D3:~$
```

lspci:

List installed PCI devices.

Input: - lspci

Output: -



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:

Output CPU architecture and characteristics.

Input: - lscpu

Output: -

df:

Show disk space usage and availability.

Input: -df

Output: -

```
user@LAPTOP-6IN5E1D3:~$ df
Filesystem      1K-blocks      Used  Available Use% Mounted on
none              4008568         4    4008564   1% /mnt/wsl
none            322598024 249724504   72873520  78% /usr/lib/wsl/drivers
none              4008568         0    4008568   0% /usr/lib/modules
none              4008568         0    4008568   0% /usr/lib/modules/5.15.146.1-microsoft-standard-WSL2
/dev/sdc       1055762868   1561820 1000497576   1% /
none              4008568        84    4008484   1% /mnt/wslg
none              4008568         0    4008568   0% /usr/lib/wsl/lib
rootfs            4005312      1884    4003428   1% /init
none              4008568       804    4007764   1% /run
none              4008568         0    4008568   0% /run/lock
none              4008568         0    4008568   0% /run/shm
tmpfs                4096         0       4096   0% /sys/fs/cgroup
none              4008568        76    4008492   1% /mnt/wslg/versions.txt
none              4008568        76    4008492   1% /mnt/wslg/doc
C:\             322598024 249724504   72873520  78% /mnt/c
D:\             675838972 321099964  354739008  48% /mnt/d
G:\              15728640   1199208   14529432   8% /mnt/g
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             40064     40064          0 100% /snap/snapd/21184
snapfuse            134272    134272          0 100% /snap/ubuntu-desktop-installer/1276
snapfuse            134912    134912          0 100% /snap/ubuntu-desktop-installer/1286
user@LAPTOP-6IN5E1D3:~$
```

dmidecode:

Decode and display DMI table information.

Input: - dmidecode

Output: -

```
user@LAPTOP-6IN5E1D3:~$ dmidecode
# dmidecode 3.3
Scanning /dev/mem for entry point.
/dev/mem: Permission denied
user@LAPTOP-6IN5E1D3:~$ sudo dmidecode
[sudo] password for user:
# dmidecode 3.3
Scanning /dev/mem for entry point.
# No SMBIOS nor DMI entry point found, sorry.
user@LAPTOP-6IN5E1D3:~$
```

ip a:

Show IP addresses and network information for all interfaces.

Input: - ip a

Output: -

```
user@LAPTOP-6IN5E1D3:~$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000
    link/ether 00:15:5d:25:86:ef brd ff:ff:ff:ff:ff:ff
    inet 172.27.192.178/20 brd 172.27.207.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::215:5dff:fe25:86ef/64 scope link
       valid_lft forever preferred_lft forever
user@LAPTOP-6IN5E1D3:~$
```

top:

Display real-time system resource usage.

Input: -top

Output: -

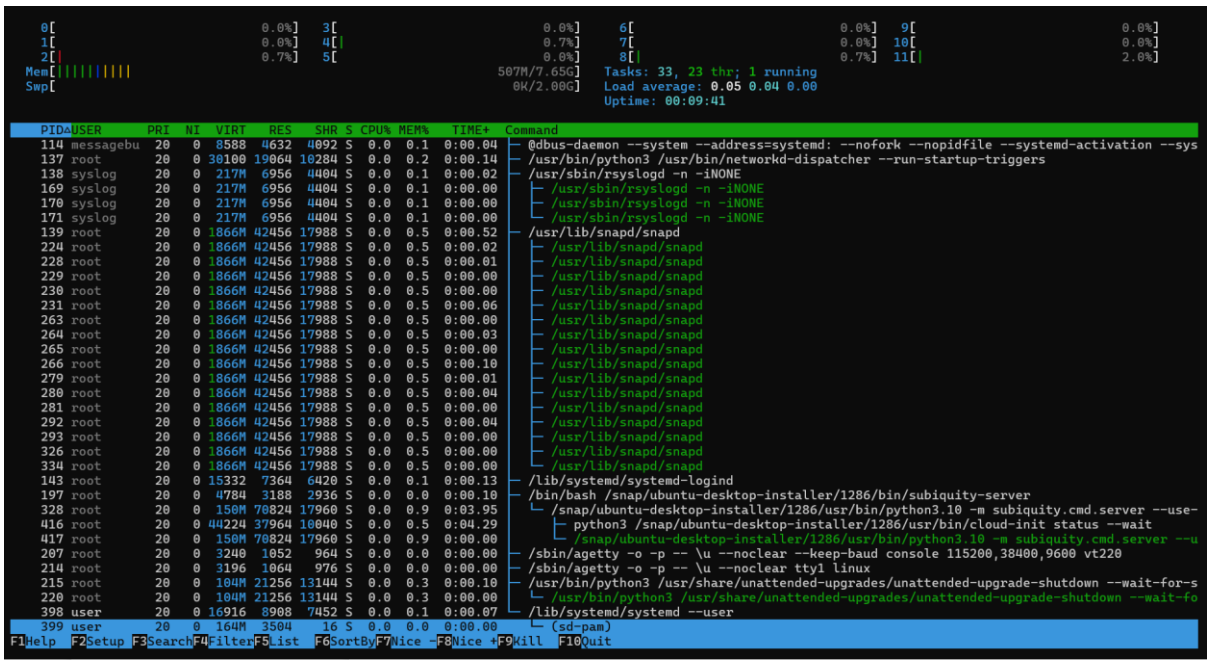
```
user@LAPTOP-6IN5E1D3:~$ top
top - 21:12:15 up 9 min,  1 user,  load average: 0.08, 0.04, 0.01
Tasks:  33 total,   1 running,  32 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.2 us,  0.1 sy,  0.0 ni, 99.7 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
MiB Mem :   7829.2 total,   6943.5 free,    505.5 used,    380.3 buff/cache
MiB Swap:   2048.0 total,   2048.0 free,      0.0 used.   7095.3 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
      1 root      20   0  165976  11108   8112 S   1.7   0.1   0:06.39 systemd
    416 root      20   0   44224  37964  10040 S   0.3   0.5   0:04.16 python3
      2 root      20   0    2280   1304   1188 S   0.0   0.0   0:00.00 init-systemd(Ub
      8 root      20   0    2280      4      0 S   0.0   0.0   0:00.00 init
     37 root      19  -1   47740  14920  13916 S   0.0   0.2   0:00.15 systemd-journal
     61 root      20   0   21976   5844   4492 S   0.0   0.1   0:00.24 systemd-udevd
     69 root      20   0    4496    144      0 S   0.0   0.0   0:00.00 snapfuse
     70 root      20   0    4764   1780   1244 S   0.0   0.0   0:01.14 snapfuse
     74 root      20   0    4628    196     48 S   0.0   0.0   0:00.00 snapfuse
     81 root      20   0    4496    160     12 S   0.0   0.0   0:00.00 snapfuse
     84 root      20   0    4744   1800   1284 S   0.0   0.0   0:03.52 snapfuse
     88 root      20   0    4496    184     36 S   0.0   0.0   0:00.00 snapfuse
     89 root      20   0    4744   1808   1312 S   0.0   0.0   0:01.68 snapfuse
     96 systemd+  20   0   25540  12676   8480 S   0.0   0.2   0:00.20 systemd-resolve
    113 root      20   0    4308   2744   2512 S   0.0   0.0   0:00.00 cron
    114 message+  20   0    8588   4632   4092 S   0.0   0.1   0:00.04 dbus-daemon
    137 root      20   0   30100  19064  10284 S   0.0   0.2   0:00.14 networkd-dispat
    138 syslog    20   0  222404   6956   4404 S   0.0   0.1   0:00.02 rsyslogd
    139 root      20   0 1911120  42456  17988 S   0.0   0.5   0:00.52 snapd
    143 root      20   0   15332   7364   6420 S   0.0   0.1   0:00.13 systemd-logind
    197 root      20   0    4784   3188   2936 S   0.0   0.0   0:00.10 subiquity-serve
    207 root      20   0    3240   1052    964 S   0.0   0.0   0:00.00 agetty
    214 root      20   0    3196   1064    976 S   0.0   0.0   0:00.00 agetty
    215 root      20   0  107224  21256  13144 S   0.0   0.3   0:00.10 unattended-upgr
    328 root      20   0  154620  70824  17960 S   0.0   0.9   0:03.92 python3.10
    364 root      20   0    7528   4924   4000 S   0.0   0.1   0:00.01 login
    398 user      20   0   16916   8908   7452 S   0.0   0.1   0:00.07 systemd
    399 user      20   0  168900   3504     16 S   0.0   0.0   0:00.00 (sd-pam)
    404 user      20   0    6124   4916   3320 S   0.0   0.1   0:00.01 bash
   1548 root      20   0    2296    120      0 S   0.0   0.0   0:00.00 SessionLeader
   1549 root      20   0    2296    128      0 S   0.0   0.0   0:00.01 Relay(1555)
   1555 user      20   0    6212   5136   3412 S   0.0   0.1   0:00.07 bash
   2597 user      20   0    7792   3720   3120 R   0.0   0.0   0:00.00 top
```

htop:

Interactive version of top with enhanced visualization.

Input: -htop

Output: -

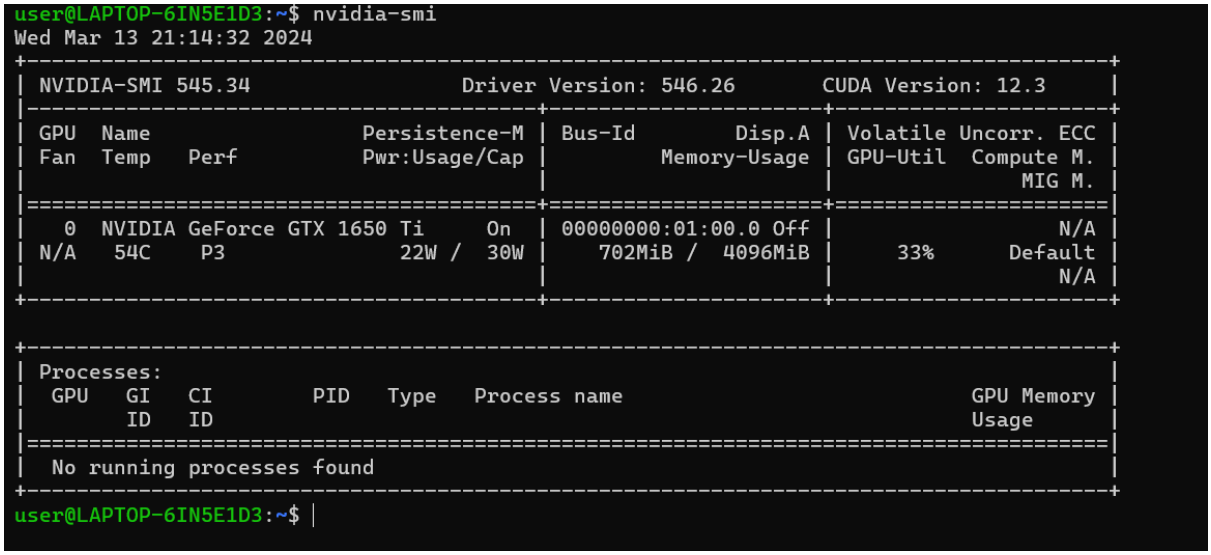


nvidia-smi:

Provide management for NVIDIA GPU devices.

Input: - nvidia-smi

Output: -

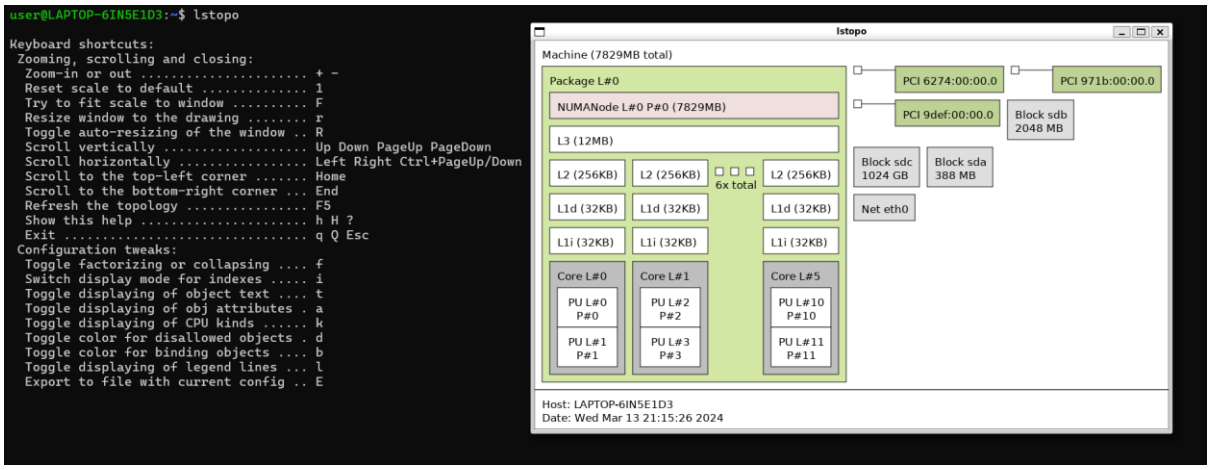


lstopo:

Generate graphical system topology representation.

Input: -lstopo

Output: -



perf:

Collect and analyze performance data.

Input: -perf

Output: -

numactl:

Control and monitor NUMA policy on NUMA systems.

Input: - numactl

Output: -

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

sar:

Collect and analyze system activity data over time.

Input: - sar -u 1 10

Output: -

```
user@LAPTOP-6IN5E1D3:~$ sar -u 1 10
Linux 5.15.146.1-microsoft-standard-WSL2 (LAPTOP-6IN5E1D3)      03/13/24        _x86_64_        (12 CPU)

22:06:39        CPU     %user     %nice   %system   %iowait    %steal     %idle
22:06:40        all      0.08      0.00      0.17      0.00      0.00     99.75
22:06:41        all      0.08      0.00      0.08      0.00      0.00     99.83
22:06:42        all      0.00      0.00      0.17      0.00      0.00     99.83
22:06:43        all      0.08      0.00      0.00      0.00      0.00     99.92
22:06:44        all      0.17      0.00      0.00      0.00      0.00     99.83
22:06:45        all      0.00      0.00      0.00      0.00      0.00    100.00
22:06:46        all      0.00      0.00      0.00      0.00      0.00    100.00
22:06:47        all      0.08      0.00      0.00      0.00      0.00     99.92
22:06:48        all      0.08      0.00      0.17      0.00      0.00     99.75
22:06:49        all      0.08      0.00      0.00      0.00      0.00     99.92
Average:        all      0.07      0.00      0.06      0.00      0.00     99.88
user@LAPTOP-6IN5E1D3:~$
```

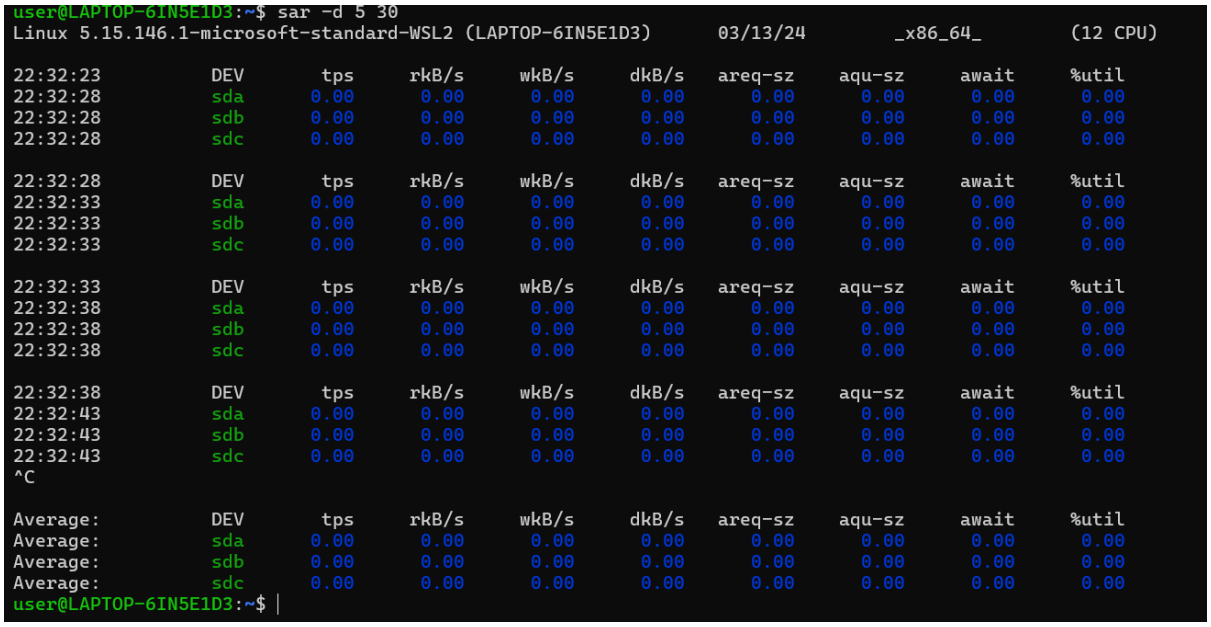
Input: - sar -r 60 5

Output: -

```
user@LAPTOP-6IN5E1D3:~$ sar -r 60 5
Linux 5.15.146.1-microsoft-standard-WSL2 (LAPTOP-6IN5E1D3)      03/13/24        _x86_64_        (12 CPU)

22:29:00    kbmemfree   kbavail kbmemused  %memused kbbuffers  kbcached  kbcommit   %commit  kbactive   kbinact   kbdirty
22:30:00      7070956   7236824    492196      6.14     13732    355448    844528      8.35     99232    512988         0
22:31:00      7070264   7236132    492856      6.15     13732    355448    845600      8.36     99240    513400         0
22:32:00      7073932   7239812    489144      6.10     13740    355448    842768      8.33     99244    511808         0
22:33:00      7074040   7239920    489008      6.10     13740    355448    841828      8.32     99236    511404         0
22:34:00      7079396   7245228    483644      6.03     13740    355448    838996      8.30     99188    509576         0
Average:      7073718   7239583    489370      6.10     13737    355448    842744      8.33     99228    511835         0
user@LAPTOP-6IN5E1D3:~$
```

Input: - sar -d 5 30

Output: -



all_in_mac

March 28, 2024

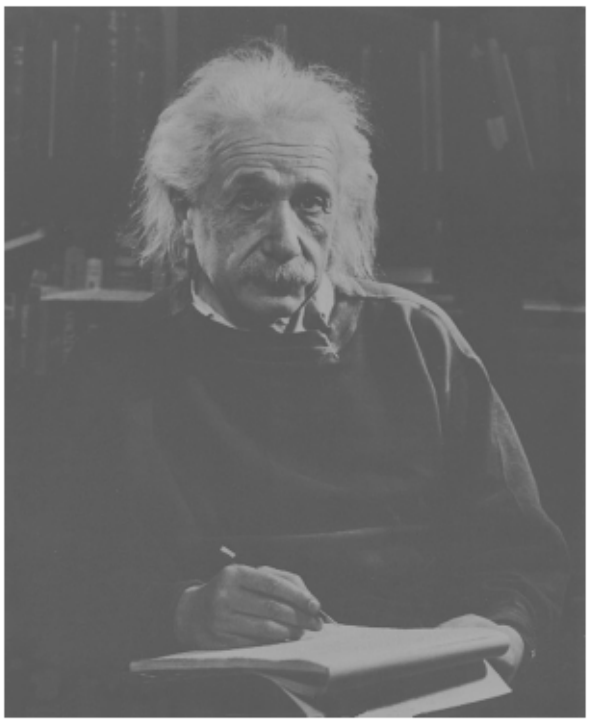
1 Image Filtering

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

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

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

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

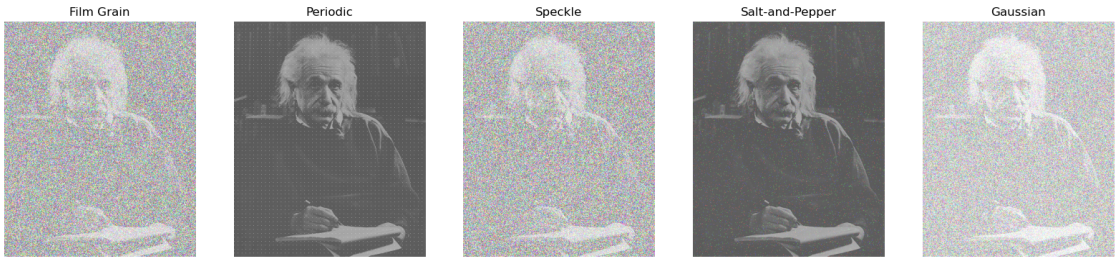


1.1 Image nosing

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

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

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

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

```
        return cv2.add(image, salt_pepper_noise)

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

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

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

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



1.2 Linear Filtering

1.2.1 Box Filter

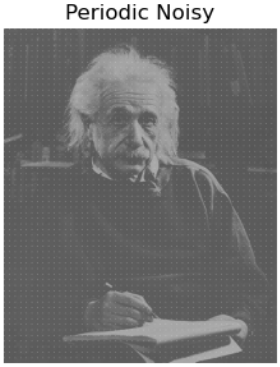
[6]:
```
def apply_box_filter(image):
    return cv2.boxFilter(image, -1, (5, 5))

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

for noise_type, noisy_image in noisy_images.items():
```

```
    plt.subplot(len(noisy_images), 2, 2*(list(noisy_images.keys()).
 ↪index(noise_type)) + 1)
    plt.imshow(cv2.cvtColor(noisy_image, cv2.COLOR_BGR2RGB))
    plt.title(noise_type + ' Noisy', fontsize=12)
    plt.axis('off')

    box_filtered = apply_box_filter(noisy_image)

    plt.subplot(len(noisy_images), 2, 2*(list(noisy_images.keys()).
 ↪index(noise_type)) + 2)
    plt.imshow(cv2.cvtColor(box_filtered, cv2.COLOR_BGR2RGB))
    plt.title(noise_type + ' Box Filtered', fontsize=12)
    plt.axis('off')

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

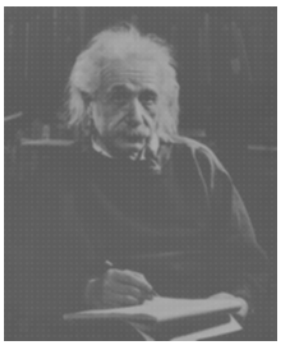
Film Grain Noisy

Film Grain Box Filtered

Periodic Noisy



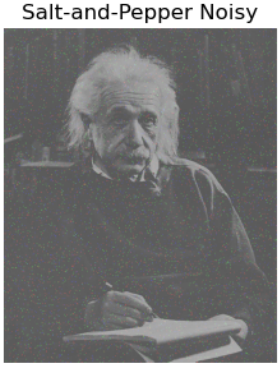
Periodic Box Filtered



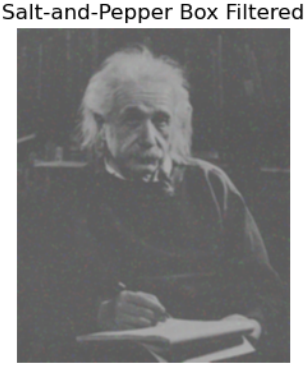
Speckle Noisy

Speckle Box Filtered

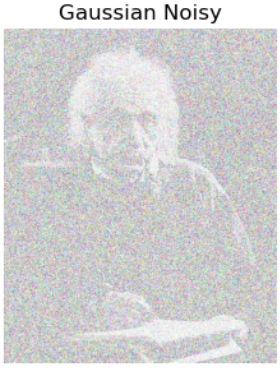
Salt-and-Pepper Noisy



Salt-and-Pepper Box Filtered



Gaussian Noisy



Gaussian Box Filtered

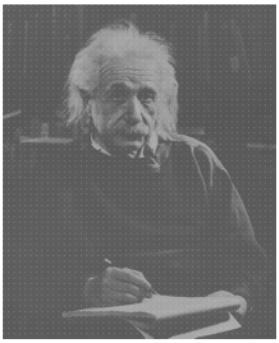
1.2.2 Gaussian Filter

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

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

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

    gaussian_filtered = apply_gaussian_filter(noisy_image)

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

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

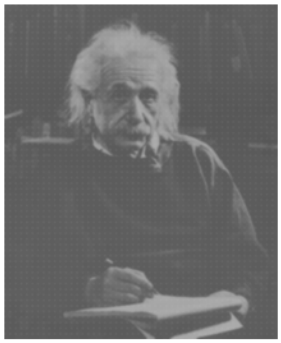
Film Grain Noisy

Film Grain Gaussian Filtered

Periodic Noisy



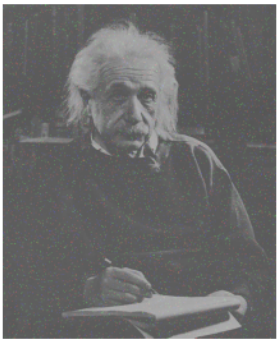
Periodic Gaussian Filtered



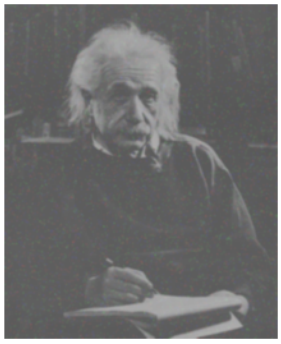
Speckle Noisy

Speckle Gaussian Filtered

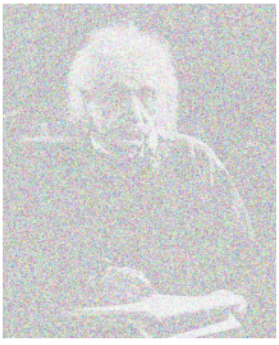
Salt-and-Pepper Noisy



Salt-and-Pepper Gaussian Filtered



Gaussian Noisy



Gaussian Gaussian Filtered

1.2.3 Sobel

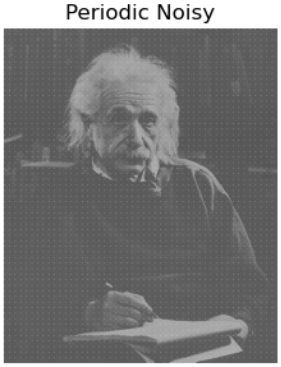
[8]:
```
if original_image is None:
    print("Error: Unable to load the image.")
else:
    sobel_x = cv2.Sobel(original_image, cv2.CV_64F, 1, 0, ksize=3)
    sobel_y = cv2.Sobel(original_image, cv2.CV_64F, 0, 1, ksize=3)

    sobel_mag = np.sqrt(sobel_x**2 + sobel_y**2)

    threshold = 38
    sobel_edges = np.where(sobel_mag > threshold, 255, 0).astype(np.uint8)

    plt.figure(figsize=(10, 5))
    plt.subplot(1, 2, 1)
    plt.imshow(original_image, cmap='gray')
    plt.title('Original Image')
    plt.axis('off')

    plt.subplot(1, 2, 2)
    plt.imshow(sobel_edges, cmap='gray')
    plt.title('Sobel Edges')
    plt.axis('off')

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

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

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

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

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

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

Film Grain Noisy

Film Grain Edges

Periodic Noisy

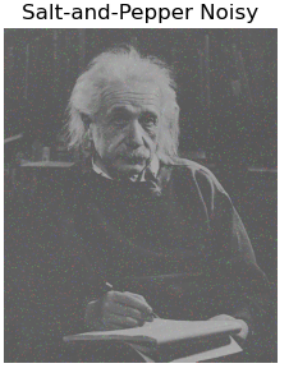


Periodic Edges

Speckle Noisy

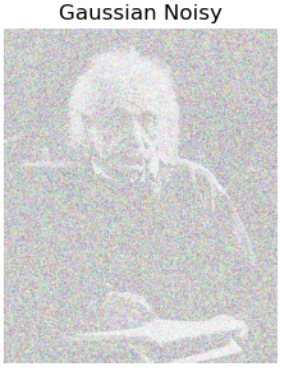
Speckle Edges

Salt-and-Pepper Noisy



Salt-and-Pepper Edges

Gaussian Noisy



Gaussian Edges

1.3 Non-linear Filtering

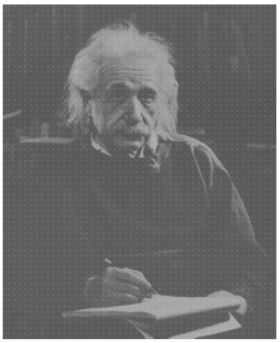
1.3.1 Median Filter

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

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

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

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

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

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

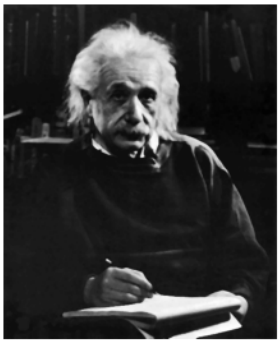
Film Grain Noisy

Film Grain Median Filtered

Periodic Noisy



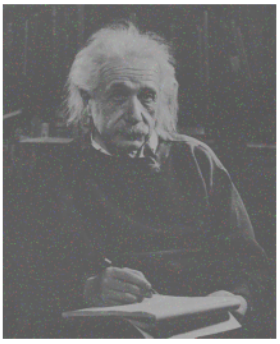
Periodic Median Filtered



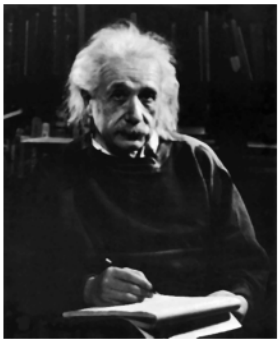
Speckle Noisy

Speckle Median Filtered

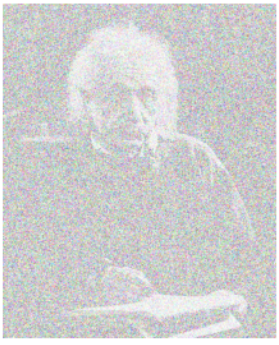
Salt-and-Pepper Noisy



Salt-and-Pepper Median Filtered



Gaussian Noisy



Gaussian Median Filtered

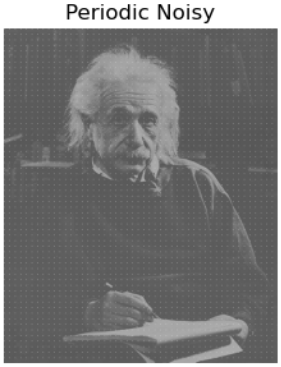
1.3.2 Rank Filter

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

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

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

    return filtered_image

num_noises = len(noisy_images)
num_images = num_noises

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

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

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

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

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

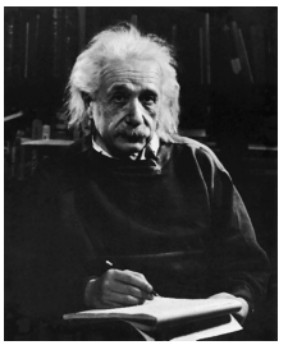
Film Grain Noisy

Film Grain Rank Filtered

Periodic Noisy



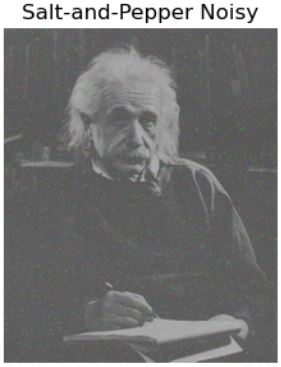
Periodic Rank Filtered



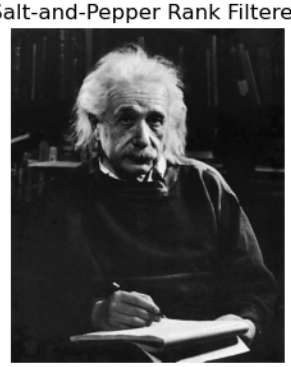
Speckle Noisy

Speckle Rank Filtered

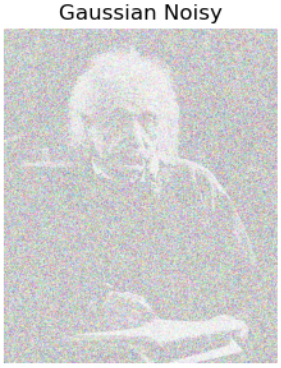
Salt-and-Pepper Noisy



Salt-and-Pepper Rank Filtered



Gaussian Noisy



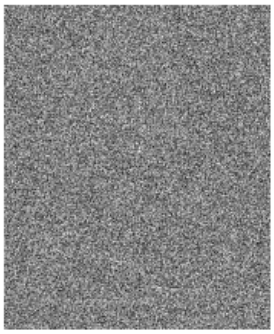
Gaussian Rank Filtered

1.3.3 Adaptive Filter

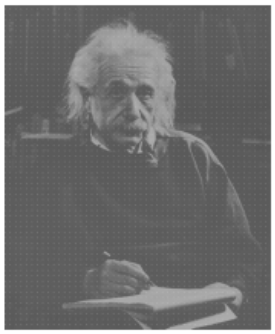
[12]:
```
def apply_adaptive_filter(image):
    gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    return cv2.adaptiveThreshold(gray_image, 255, cv2.
 ↪ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY, 11, 2)

num_noises = len(noisy_images)
num_images = num_noises + 1

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

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

    adaptive_filtered = apply_adaptive_filter(noisy_image)

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

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

Film Grain Noisy

Film Grain Adaptive Filtered



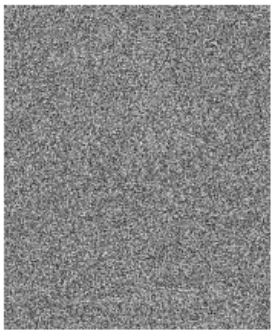
Periodic Noisy



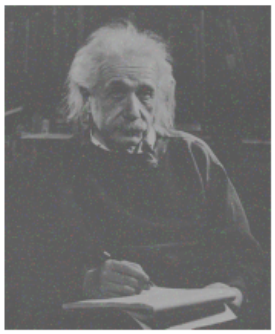
Periodic Adaptive Filtered

Speckle Noisy

Speckle Adaptive Filtered



Salt-and-Pepper Noisy



Salt-and-Pepper Adaptive Filtered

Gaussian Noisy

Gaussian Adaptive Filtered

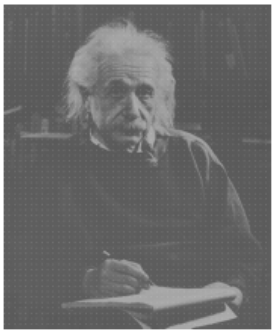
1.3.4 Bilateral Filter

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

num_noises = len(noisy_images)
num_images = num_noises + 1

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

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

    bilateral_filtered = apply_bilateral_filter(noisy_image)

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

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

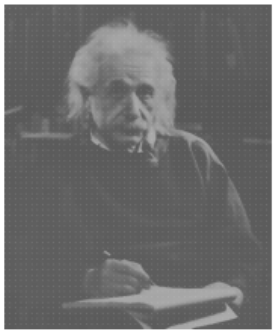
Film Grain Noisy

Film Grain Bilateral Filtered

Periodic Noisy



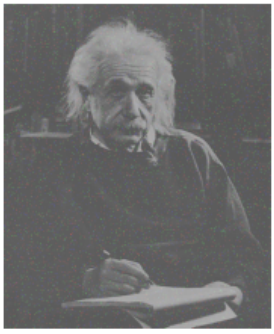
Periodic Bilateral Filtered



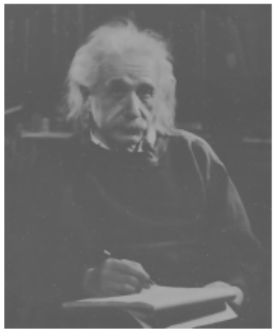
Speckle Noisy

Speckle Bilateral Filtered

Salt-and-Pepper Noisy



Salt-and-Pepper Bilateral Filtered



Gaussian Noisy

Gaussian Bilateral Filtered

2 Image Filtering on colour image

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

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

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

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

```
        return cv2.add(image, periodic_noise)

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

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

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

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

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

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

2.1 Linear Filtering

2.1.1 BOX Filter

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

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

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

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

    box_filtered = apply_box_filter(noisy_image)

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

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

Film Grain Noisy

Film Grain Box Filtered

Periodic Noisy

Periodic Box Filtered

Speckle Noisy

Speckle Box Filtered

Salt-and-Pepper Noisy

Salt-and-Pepper Box Filtered

Gaussian Noisy

Gaussian Box Filtered

2.1.2 Gaussian Filter

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

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

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

    gaussian_filtered = apply_gaussian_filter(noisy_image)

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

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

Film Grain Noisy

Film Grain Gaussian Filtered

Periodic Noisy

Periodic Gaussian Filtered

Speckle Noisy

Speckle Gaussian Filtered

Salt-and-Pepper Noisy

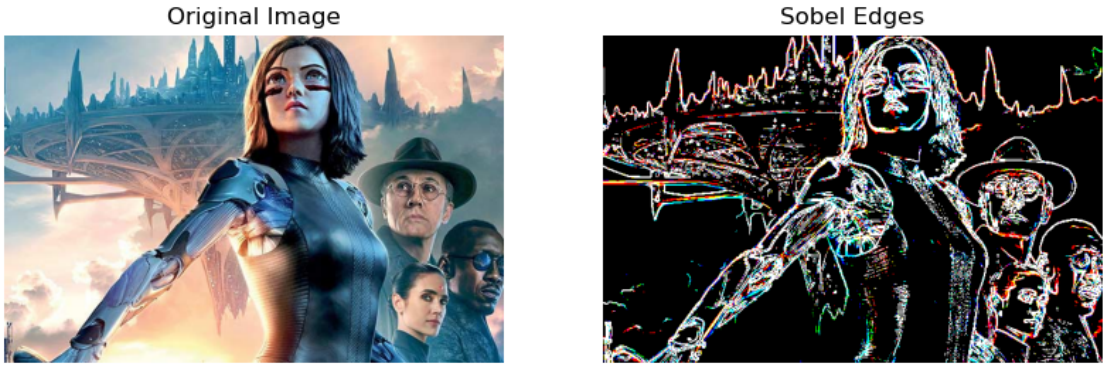
Salt-and-Pepper Gaussian Filtered

Gaussian Noisy

Gaussian Gaussian Filtered

2.1.3 Sobel

[20]:
```
if original_image is None:
    print("Error: Unable to load the image.")
else:
    sobel_x = cv2.Sobel(original_image, cv2.CV_64F, 1, 0, ksize=3)
    sobel_y = cv2.Sobel(original_image, cv2.CV_64F, 0, 1, ksize=3)

    sobel_mag = np.sqrt(sobel_x**2 + sobel_y**2)

    threshold = 130
    sobel_edges = np.where(sobel_mag > threshold, 255, 0).astype(np.uint8)

    plt.figure(figsize=(10, 5))
    plt.subplot(1, 2, 1)
    plt.imshow(original_image, cmap='gray')
    plt.title('Original Image')
    plt.axis('off')

    plt.subplot(1, 2, 2)
    plt.imshow(sobel_edges, cmap='gray')
    plt.title('Sobel Edges')
    plt.axis('off')
    plt.savefig('sobel_filter_output.png', bbox_inches='tight')
    plt.show()
```

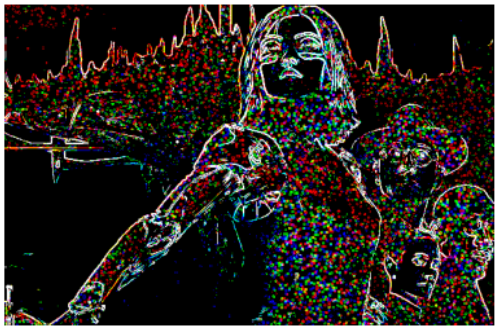


[21]:
```
def detect_edges(image):
    sobel_x = cv2.Sobel(image, cv2.CV_64F, 1, 0, ksize=3)
    sobel_y = cv2.Sobel(image, cv2.CV_64F, 0, 1, ksize=3)
    sobel_mag = np.sqrt(sobel_x**2 + sobel_y**2)
```

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

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

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

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

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

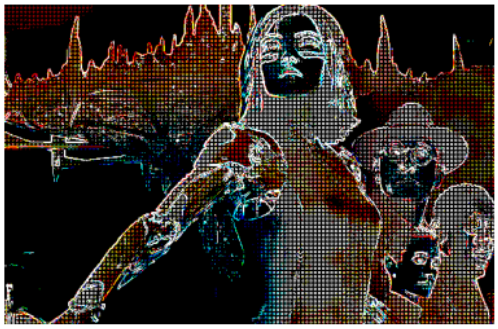
Film Grain Noisy

Film Grain Edges



Periodic Noisy

Periodic Edges



Speckle Noisy

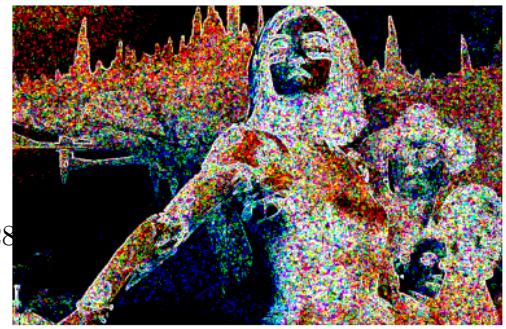
Speckle Edges

Salt-and-Pepper Noisy

Salt-and-Pepper Edges

Gaussian Noisy

Gaussian Edges



2.2 Non-linear Filtering

2.2.1 Median Filter

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

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

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

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

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

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

Film Grain Noisy

Film Grain Median Filtered

Periodic Noisy

Periodic Median Filtered

Speckle Noisy

Speckle Median Filtered

Salt-and-Pepper Noisy

Salt-and-Pepper Median Filtered

Gaussian Noisy

Gaussian Median Filtered

2.2.2 Rank Filter

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

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

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

    return filtered_image

num_noises = len(noisy_images)
num_images = num_noises

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

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

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

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

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

Film Grain Noisy

Film Grain Rank Filtered

Periodic Noisy

Periodic Rank Filtered

Speckle Noisy

Speckle Rank Filtered

Salt-and-Pepper Noisy

Salt-and-Pepper Rank Filtered

Gaussian Noisy

Gaussian Rank Filtered

2.2.3 Adaptive Filter

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

num_noises = len(noisy_images)
num_images = num_noises + 1

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

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

    adaptive_filtered = apply_adaptive_filter(noisy_image)

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

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

Film Grain Noisy

Film Grain Adaptive Filtered

Periodic Noisy

Periodic Adaptive Filtered

Speckle Noisy

Speckle Adaptive Filtered

Salt-and-Pepper Noisy

Salt-and-Pepper Adaptive Filtered

Gaussian Noisy

Gaussian Adaptive Filtered

2.2.4 Bilateral Filter

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

num_noises = len(noisy_images)
num_images = num_noises + 1

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

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

    bilateral_filtered = apply_bilateral_filter(noisy_image)

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

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

Film Grain Noisy

Film Grain Bilateral Filtered

Periodic Noisy

Periodic Bilateral Filtered

Speckle Noisy

Speckle Bilateral Filtered

Salt-and-Pepper Noisy

Salt-and-Pepper Bilateral Filtered

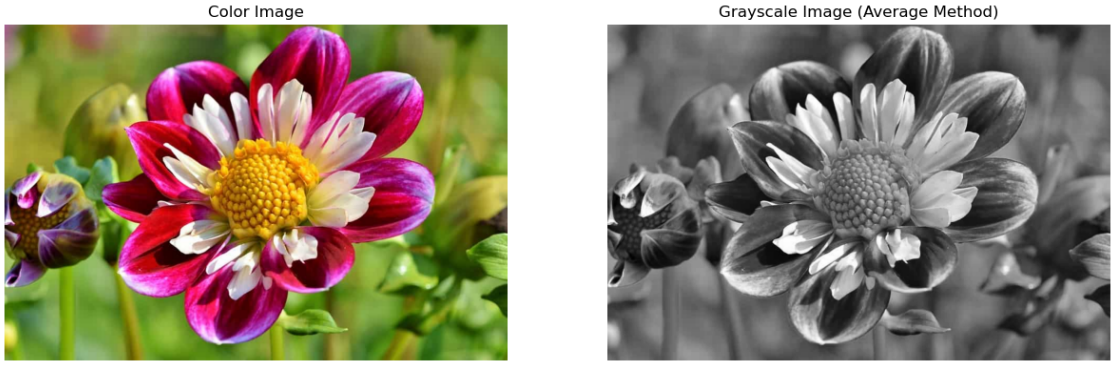
Gaussian Noisy

Gaussian Bilateral Filtered

3 Gray Scale Imaging

3.1 Average Method

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

    grayscale_img = np.uint8(grayscale_img)

    return grayscale_img

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

grayscale_img_avg = average_method(color_img)

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

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

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

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

plt.show()
```



3.2 Luminosity Method

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

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

    return grayscale_img

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

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

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

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

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

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

plt.show()
```

3.3 Single Channel Method

[28]:
```
color_img = cv2.imread("I:/My Drive/Image_filtering/images/images/animated2.
 ↪jpg")

green_channel = color_img[:,:,1]

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

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

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

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

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

plt.subplot(2,2,2)
plt.imshow(green_channel , cmap='gray')  # Specify grayscale colormap
plt.title('Grayscale Image (Single Channel - Green)')
plt.axis('off')  # Hide axis

plt.subplot(2,2,3)
plt.imshow(red_channel , cmap='gray')  # Specify grayscale colormap
plt.title('Grayscale Image (Single Channel - Red)')
plt.axis('off')  # Hide axis

plt.subplot(2,2,4)
plt.imshow(blue_channel , cmap='gray')  # Specify grayscale colormap
plt.title('Grayscale Image (Single Channel - Blue)')
plt.axis('off')  # Hide axis

plt.show()
```

4 Image Blurring

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

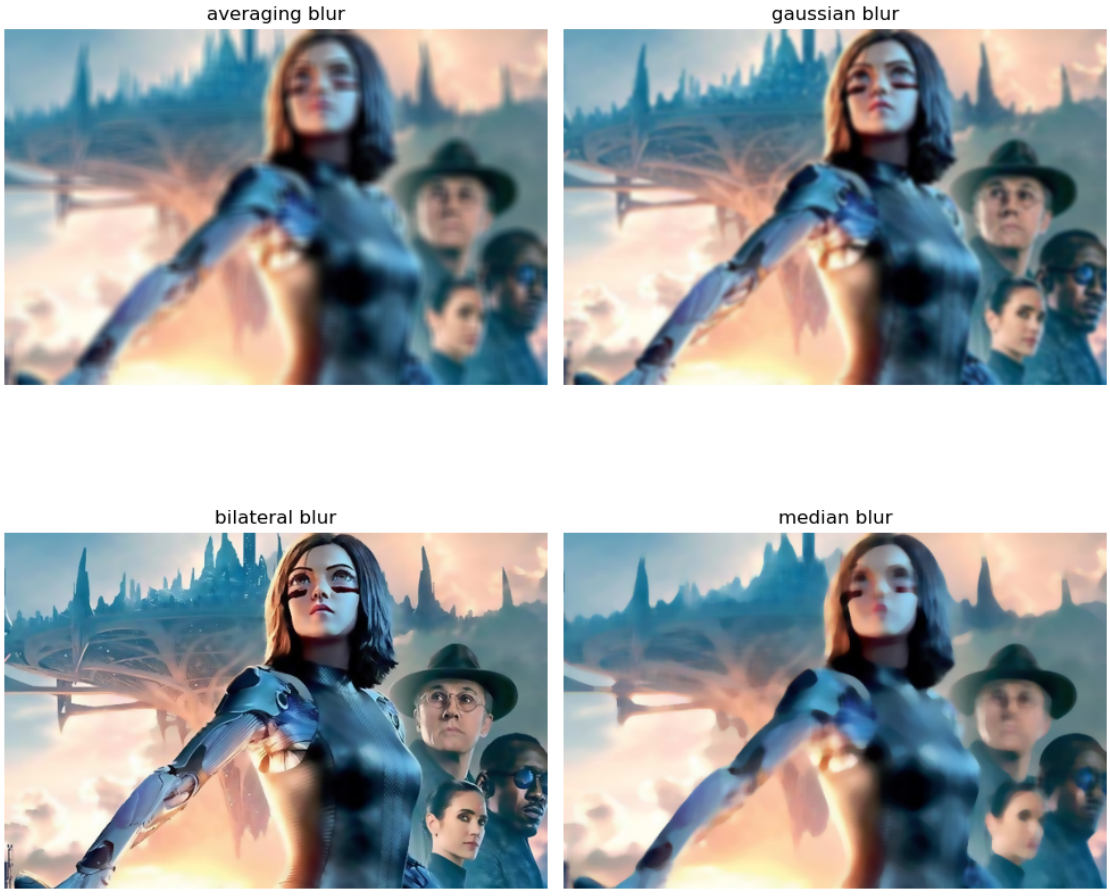
```
plt.imshow(img)
plt.axis('off')  # Hide axis
plt.show()
```

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

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

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

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

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

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

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

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

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



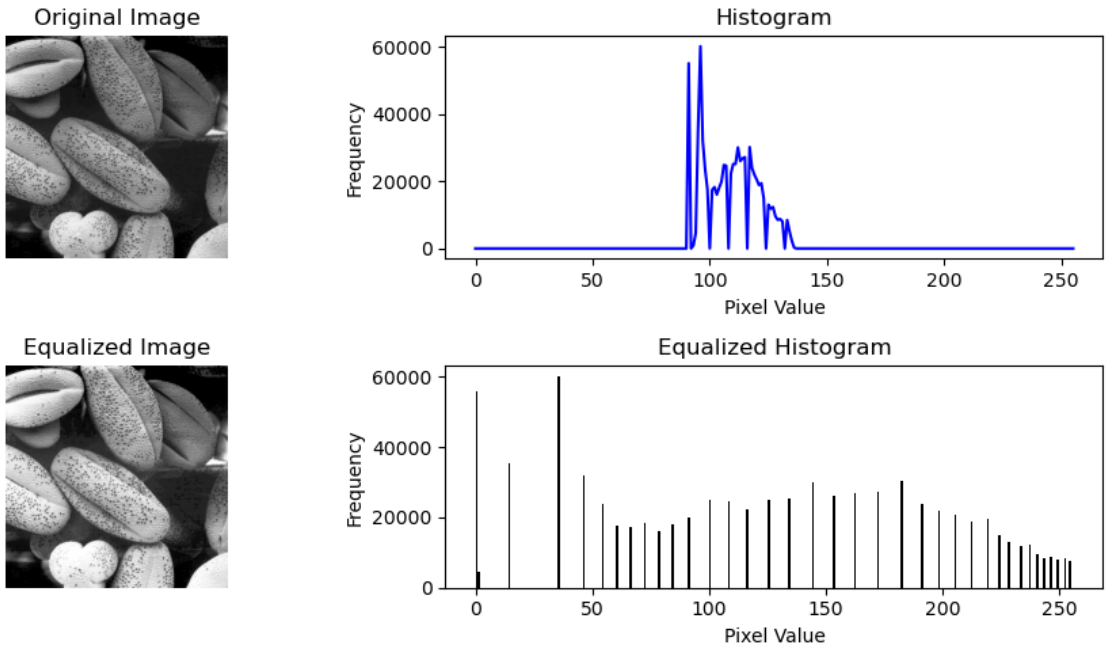
5 Histogram based image analysis

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

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

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

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

    # Apply histogram equalization
    equalized_image = cdf[image]

    return equalized_image, hist
```

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

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

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

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

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

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

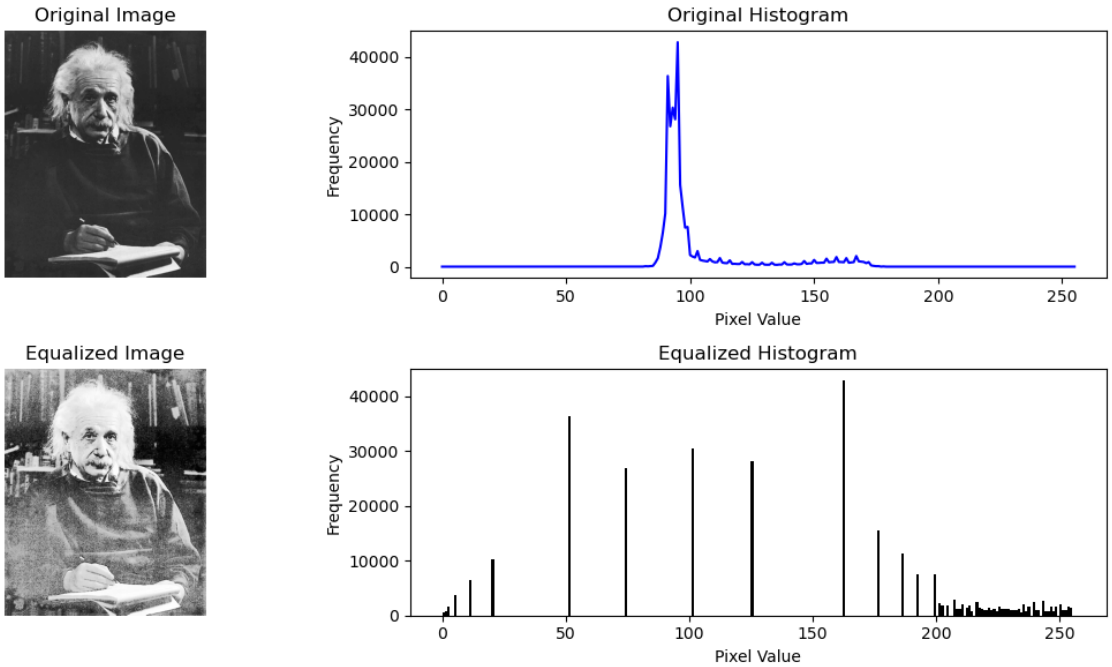


[37]:
```
image_path = "I:/My Drive/Image_filtering/images/images/
 ↪Fig0354(a)(einstein_orig).tif"
image = Image.open(image_path).convert('L')
```

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

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

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

    # Apply histogram equalization
    equalized_image = cdf[image]

    return equalized_image, hist, cdf
```

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

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

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

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

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

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

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



6 Image Scaling

6.1 Nearest neighbor

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

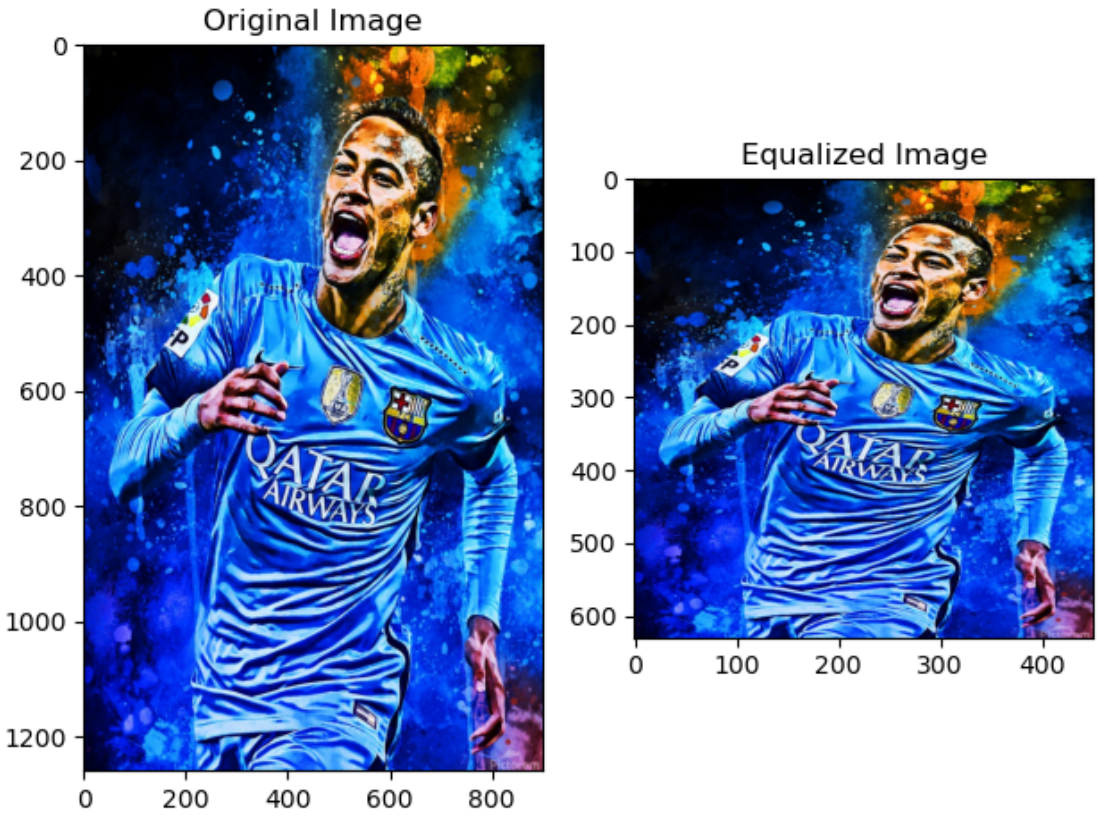
[42]:
```
# original_img = Image.open("/Users/user/Downloads/animated2.jpg")
original_img = Image.open("I:/My Drive/Image_filtering/images/images/animated2.
 ↪jpg")

original_width, original_height = original_img.size

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

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

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

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

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

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

[45]:
```
original_width, original_height = original_img.size

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

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

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

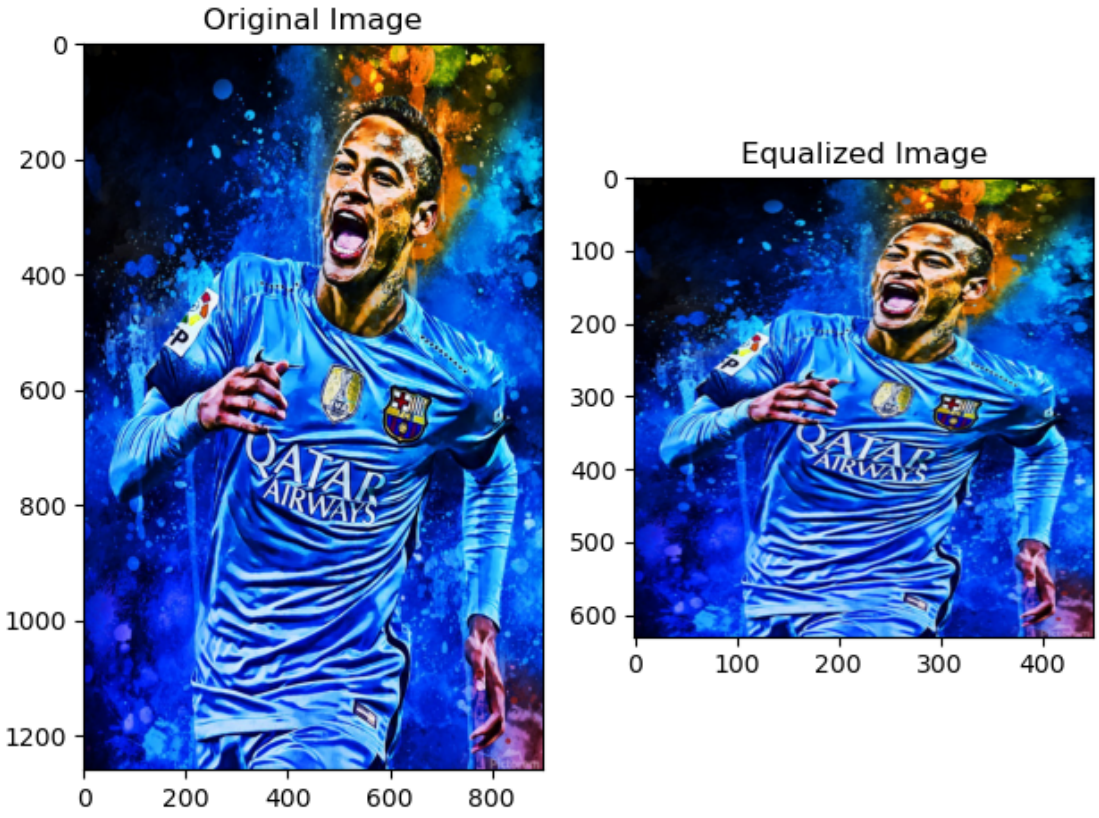


6.2 Bilinar Interpolation

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

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

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

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

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

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

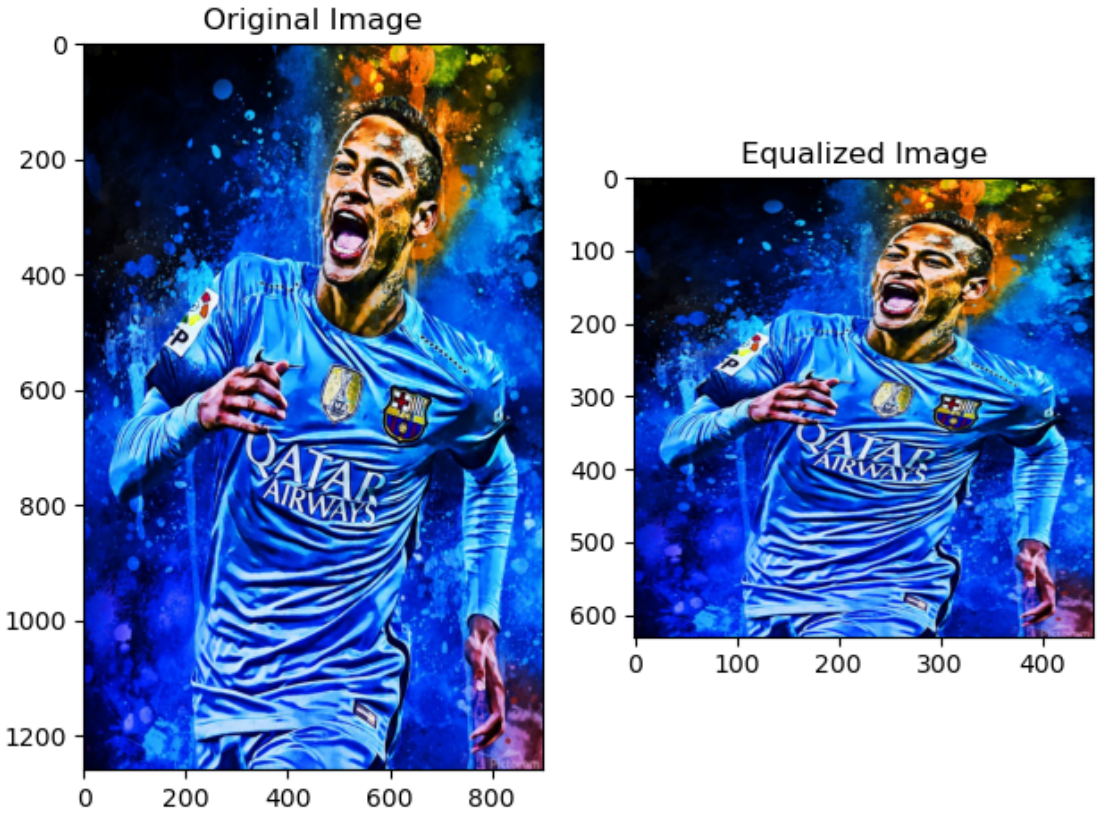


6.3 Bicubic Interpolation

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

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

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

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

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

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

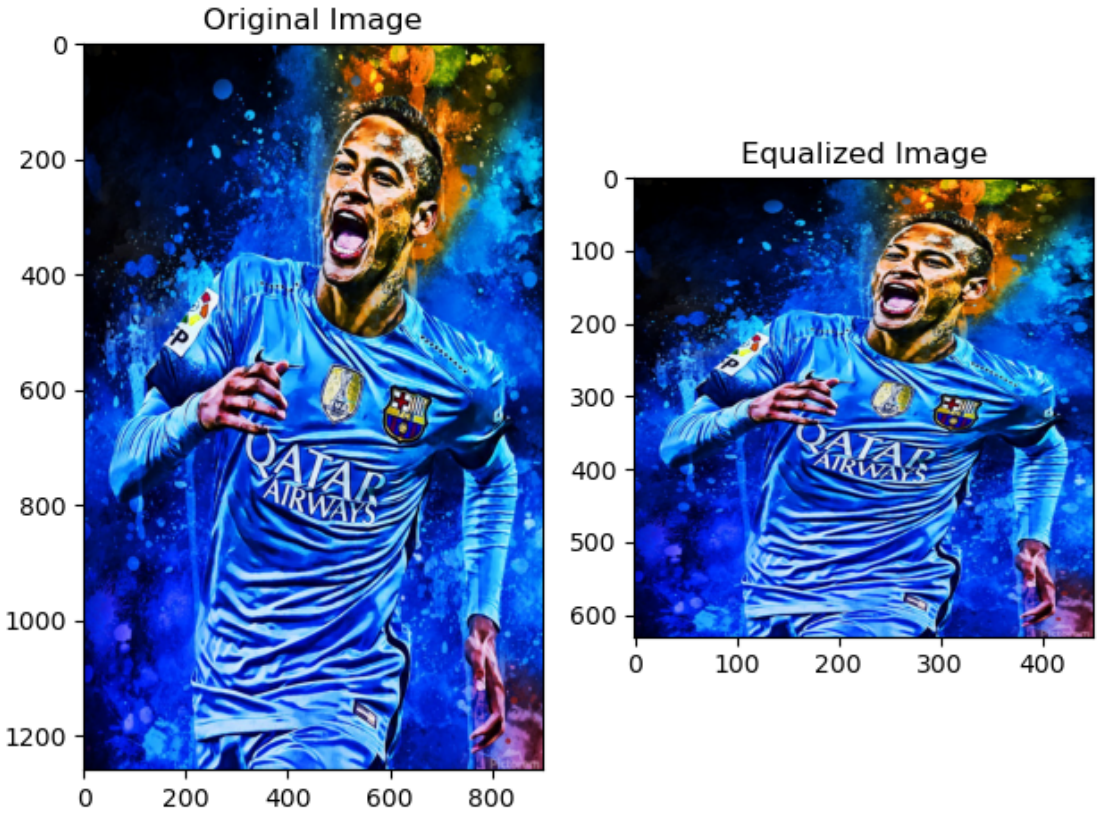


6.4 Lanczos Interpolation

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

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

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

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

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

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



Assignment 16

CUDA programming

1. **What is CUDA ?**

A. CUDA = Compute Unified Device Architecture
CUDA is a parallel computing platform and programming model developed by NVIDIA for general computing on graphical processing units (GPUs).

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

2. **What is the prerequisite for learning CUDA?**

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

3. **Which are the languages that support CUDA?**

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

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

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

5. **How CUDA works?**

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

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

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

A. Benefits :

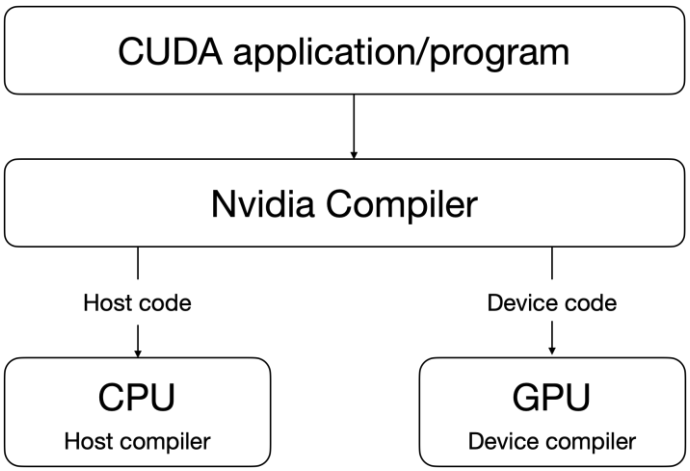
- Massive parallelism
- Performance boost
- Integrated memory and shared memory

Limitations :

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

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

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



The GPU runs the device code, whereas the CPU runs the host code. The NVCC runs a CUDA programme and isolates the host code from the device code. To do

this, specific CUDA keywords are searched for. The code that is intended to run on the GPU (device code) is identified by special CUDA keywords known as 'Kernels' that label data-parallel functions. The NVCC further compiles the device code, which is then run on the GPU.

8. **Explain CUDA thread organization.**

A.

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

A.

HPC_assignment_17

April 16, 2024

1 HPC assignment 17

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

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

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

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

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

[ ]:
```
%%writefile hello_1_1.cu

#include <stdio.h>

__global__ void cuda_hello_1_1() {
    printf("Hello World from GPU with grid dimension (1, 1) and block dimension␣
 ↪(1, 1)!\n");
}

int main() {
    cuda_hello_1_1<<<1,1>>>();
    cudaDeviceSynchronize(); // Make sure all GPU work is done before exiting
    return 0;
}
```

```
Writing hello_1_1.cu
```

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

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

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

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

1.2.1 GPU

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

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

```
/usr/local/lib/python3.10/dist-packages (from pycuda) (1.4.4)
Collecting mako (from pycuda)
  Downloading Mako-1.3.3-py3-none-any.whl (78 kB)
                            78.8/78.8 kB
10.9 MB/s eta 0:00:00
Requirement already satisfied: platformdirs>=2.2.0 in
/usr/local/lib/python3.10/dist-packages (from pytools>=2011.2->pycuda) (4.2.0)
Requirement already satisfied: typing-extensions>=4.0 in
/usr/local/lib/python3.10/dist-packages (from pytools>=2011.2->pycuda) (4.11.0)
Requirement already satisfied: MarkupSafe>=0.9.2 in
/usr/local/lib/python3.10/dist-packages (from mako->pycuda) (2.1.5)
Building wheels for collected packages: pycuda
  Building wheel for pycuda (pyproject.toml) … done
  Created wheel for pycuda: filename=pycuda-2024.1-cp310-cp310-linux_x86_64.whl
size=661204
sha256=51efb7c5582dd86e48b9404a05e0a366352406f4840bf4dc162fe9a89aa2ad1c
  Stored in directory: /root/.cache/pip/wheels/12/34/d2/9a349255a4eca3a486d82c79
d21e138ce2ccd90f414d9d72b8
Successfully built pycuda
Installing collected packages: pytools, mako, pycuda
Successfully installed mako-1.3.3 pycuda-2024.1 pytools-2024.1.1
```

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

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

[ ]:
```python
# Compile the CUDA kernel code
cuda_module = SourceModule(cuda_kernel_code)

# Get a reference to the CUDA kernel function
vector_add_cuda = cuda_module.get_function("vector_add")
```

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

    # Create device arrays
```

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

    # Copy data to device
    cuda.memcpy_htod(a_gpu, a)
    cuda.memcpy_htod(b_gpu, b)

    # Define block and grid dimensions
    block_dim = (256, 1, 1)
    grid_dim = ((n + block_dim[0] - 1) // block_dim[0], 1)

    start_time = time.time()

    # Launch the CUDA kernel
    vector_add_cuda(a_gpu, b_gpu, c_gpu, np.int32(n), block=block_dim,␣
 ↪grid=grid_dim)

    # Synchronize threads to ensure all output is calculated
    cuda.Context.synchronize()

    end_time = time.time()

    # Copy result back to host
    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)

# Perform vector addition on GPU
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 0.0007643699645996094 seconds.
Vector addition of size 20000 on GPU took 6.818771362304688e-05 seconds.
```

1.2.2 CPU

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

[ ]:
```
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)

# Perform vector addition on CPU
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 2.3365020751953125e-05 seconds.
Vector addition of size 20000 on CPU took 1.9311904907226562e-05 seconds.
```

- Vector addition of size 100 on CPU took 2.384185791015625e-05 seconds.
- Vector addition of size 20000 on CPU took 1.9788742065429688e-05 seconds.

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

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

1.3.1 GPU

[ ]:
```
def matrix_multiply_gpu(a, b):
    # Define CUDA kernel code for matrix multiplication
    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;
        }
    }
    """

    # Compile CUDA kernel code
    mod = SourceModule(cuda_code)

    # Get kernel function
    matrix_multiply_cuda = mod.get_function("matrix_multiply")

    # Allocate memory on device
    a_gpu = cuda.mem_alloc(a.nbytes)
    b_gpu = cuda.mem_alloc(b.nbytes)
    c_gpu = cuda.mem_alloc(a.nbytes)

    # Copy input matrices to device
    cuda.memcpy_htod(a_gpu, a)
    cuda.memcpy_htod(b_gpu, b)

    # Define grid and block dimensions
    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)

    # Call CUDA kernel
    matrix_multiply_cuda(a_gpu, b_gpu, c_gpu, np.int32(a.shape[0]),␣
↪block=block_size, grid=grid_size)

    # Copy result back to host
    c = np.empty_like(a)
    cuda.memcpy_dtoh(c, c_gpu)

    return c
```

[ ]:
```
# Function to generate random matrices
def generate_random_matrix(rows, cols):
    return np.random.rand(rows, cols).astype(np.float32)
```

[ ]:
```
# Function to measure time taken for matrix multiplication
def measure_time(matrix_size, func, *args):
    start_time = time.time()
    result = func(*args)
    end_time = time.time()
    return result, end_time - start_time
```

[ ]:
```
# Sizes of matrices to be multiplied
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 matrix multiplication
    gpu_result, gpu_time = measure_time(size, matrix_multiply_gpu, a, b)
    print(f"GPU time: {gpu_time:.6f} seconds")
```

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

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

1.3.2 CPU

[ ]:
```
# CPU matrix multiplication
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
```

[ ]:
```
# Function to generate random matrices
def generate_random_matrix(rows, cols):
    return np.random.rand(rows, cols).astype(np.float32)
```

[ ]:
```
# Function to measure time taken for matrix multiplication
def measure_time(matrix_size, func, *args):
    start_time = time.time()
    result = func(*args)
    end_time = time.time()
    return result, end_time - start_time
```

[ ]:
```
# Sizes of matrices to be multiplied
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)
```

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

```
Matrix size: (20, 20)
CPU time: 0.004824 seconds

Matrix size: (1024, 1024)
CPU time: 704.230331 seconds
```

- CPU Time for 1024: 0.12308359146118164 seconds
- CPU Time for 20: 0.0019140243530273438 seconds
- Matrix size: (20, 20)
- GPU time: 0.703994 seconds
- Matrix size: (1024, 1024)
- GPU time: 0.014648 seconds

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

[ ]:
```
import pycuda.driver as cuda

# Initialize PyCUDA
cuda.init()

# Get the number of CUDA devices
num_devices = cuda.Device.count()

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

# Iterate over each CUDA device and print its properties
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
```

HPC_Assignment_18

April 25, 2024

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

1.1 Gaussian Blur

1.1.1 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.

1.1.2 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

1.1.3 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/train/Cat"
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: 550
Time taken for sequential processing: 12.176469564437866 seconds
```

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

directory = "/content/drive/MyDrive/train/Cat"
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: 550
Time taken for parallel processing: 2.5575218200683594 seconds
```

1.2 FFT - Fast Fourier Transform

1.2.1 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

1.2.2 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.

1.2.3 Performance Analysis

Sequential

[ ]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy import fftpack
from PIL import Image
import os
import time
directory = "/content/drive/MyDrive/train/Cat"
num_images = 0
start_time = time.time()

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: 550
Time taken for sequential processing: 17.599610805511475 seconds
```

Parallel

[ ]:
```
import os
import time
```

```
import cupy as cp
from PIL import Image

directory = "/content/drive/MyDrive/train/Cat"

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: 550
Time taken for parallel processing: 3.021836996078491 seconds
```