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')
```

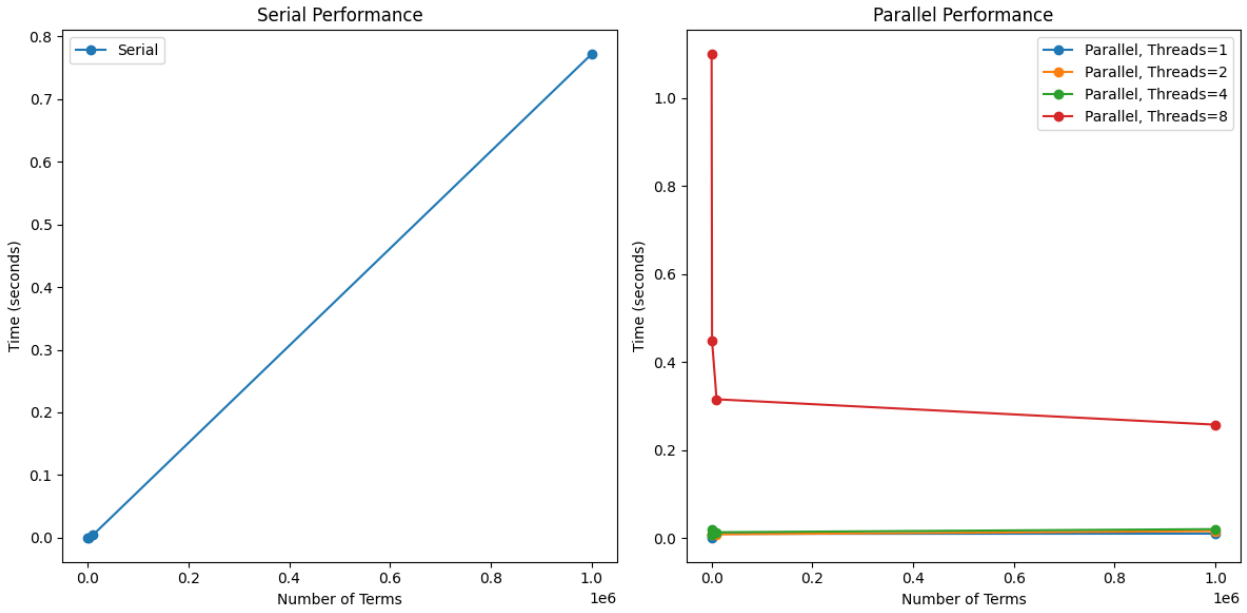
```
    plt.ylabel('Time (seconds)')
    plt.legend()

    # 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 [ ]: