Assignment - 1

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

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

Using Numpy

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

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

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

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

        concurrent.futures.wait(futures)

    return result

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

results_list = []

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

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

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

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

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

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

Using loop

In [2]:

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

In [3]:
```
def multiply_matrix(A, B, result, start_row, end_row):
    try:
        for i in range(start_row, end_row):
            for j in range(N):
                result[i, j] = 0
                for k in range(N):
                    result[i, j] += A[i, k] * B[k, j]
    except NameError as e:
        pass
```

In [4]:
```
def measure_time(matrix_size, num_threads=1):
    A = np.random.rand(matrix_size, matrix_size)
    B = np.random.rand(matrix_size, matrix_size)
    result = np.zeros((matrix_size, matrix_size))

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

    start_time = time.time()

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

    for thread in threads:
        thread.join()

    end_time = time.time()

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

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

    results = []

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

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

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

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

In [7]:

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

Out[7]:

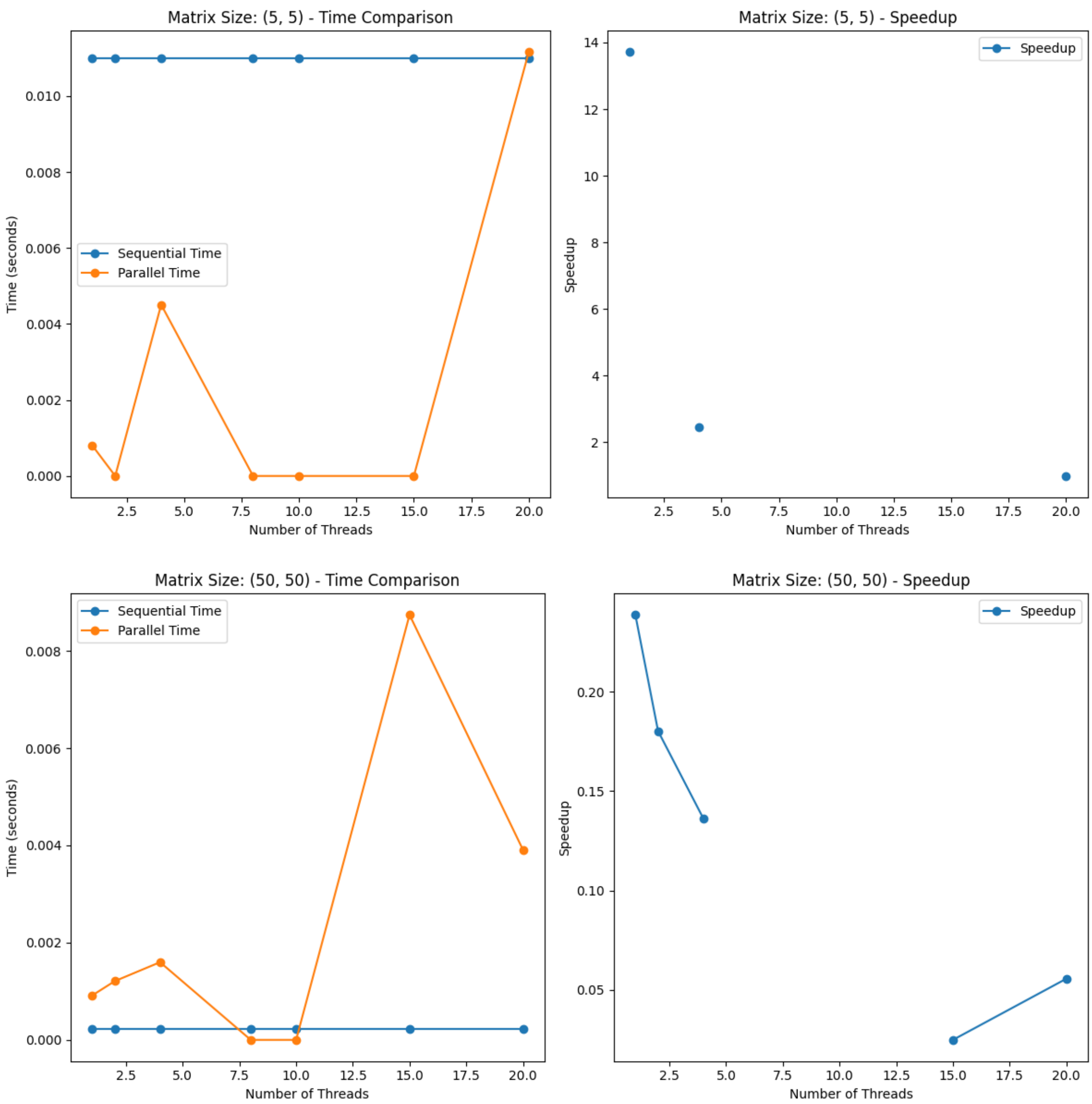
	Matrix Size	Threads	Serial Time	Parallel Time	Speedup	Efficiency
0	5	1	1.000000e-10	1.000000e-10	1.000000e+00	1.000000e+00
1	5	2	1.000000e-10	4.018307e-03	2.488610e-08	1.244305e-08
2	5	4	1.000000e-10	1.000000e-10	1.000000e+00	2.500000e-01
3	5	8	1.000000e-10	5.263329e-03	1.899938e-08	2.374923e-09
4	5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

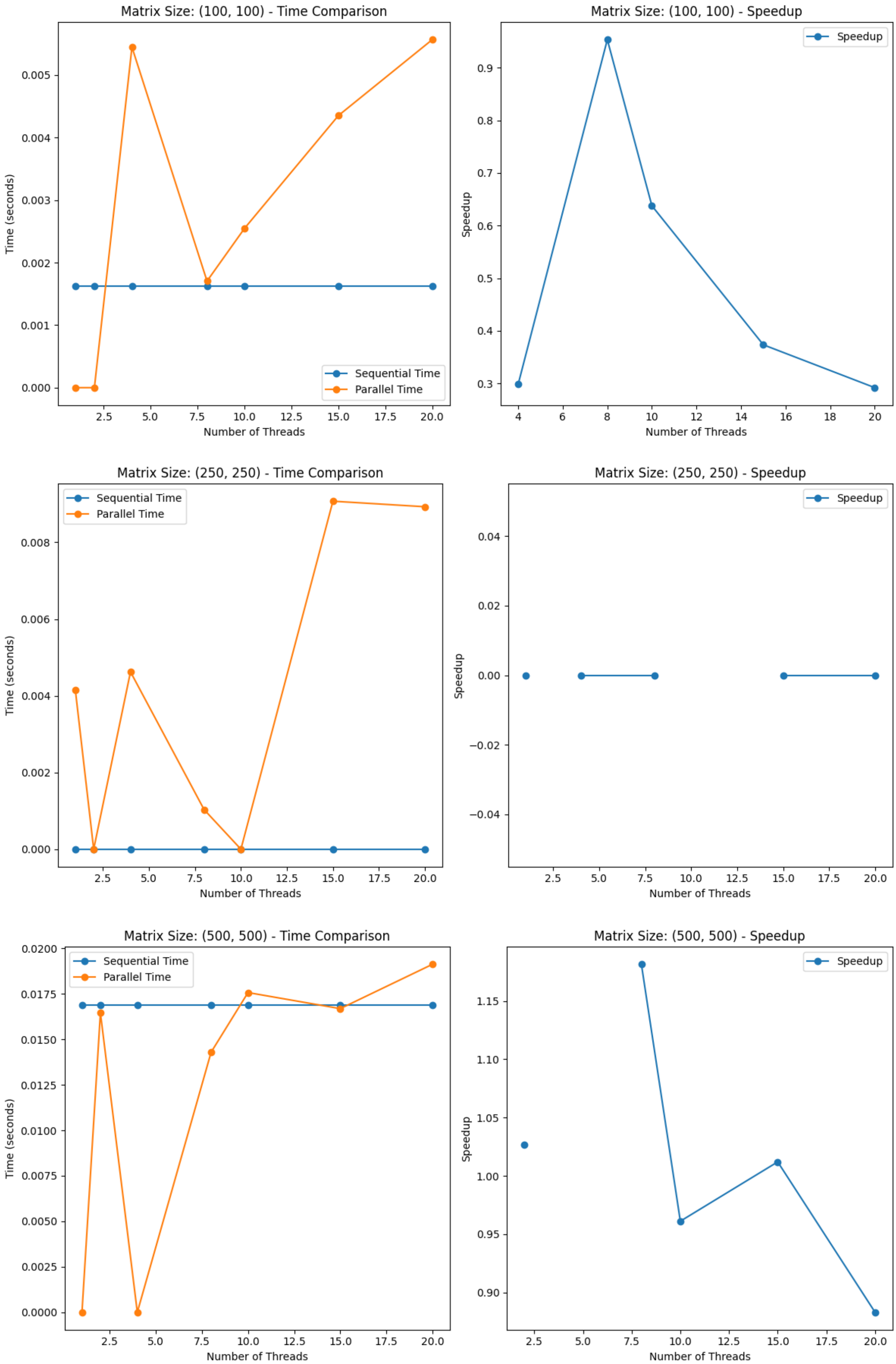
	Matrix Size	Threads	Serial Time	Parallel Time	Speedup	Efficiency
45	1000	8	1.000000e-10	1.721144e-02	5.810090e-09	7.262613e-10
46	1000	10	1.000000e-10	1.010704e-02	9.894093e-09	9.894093e-10
47	1000	15	1.000000e-10	6.479263e-03	1.543385e-08	1.028924e-09
48	1000	20	1.000000e-10	1.561546e-02	6.403909e-09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

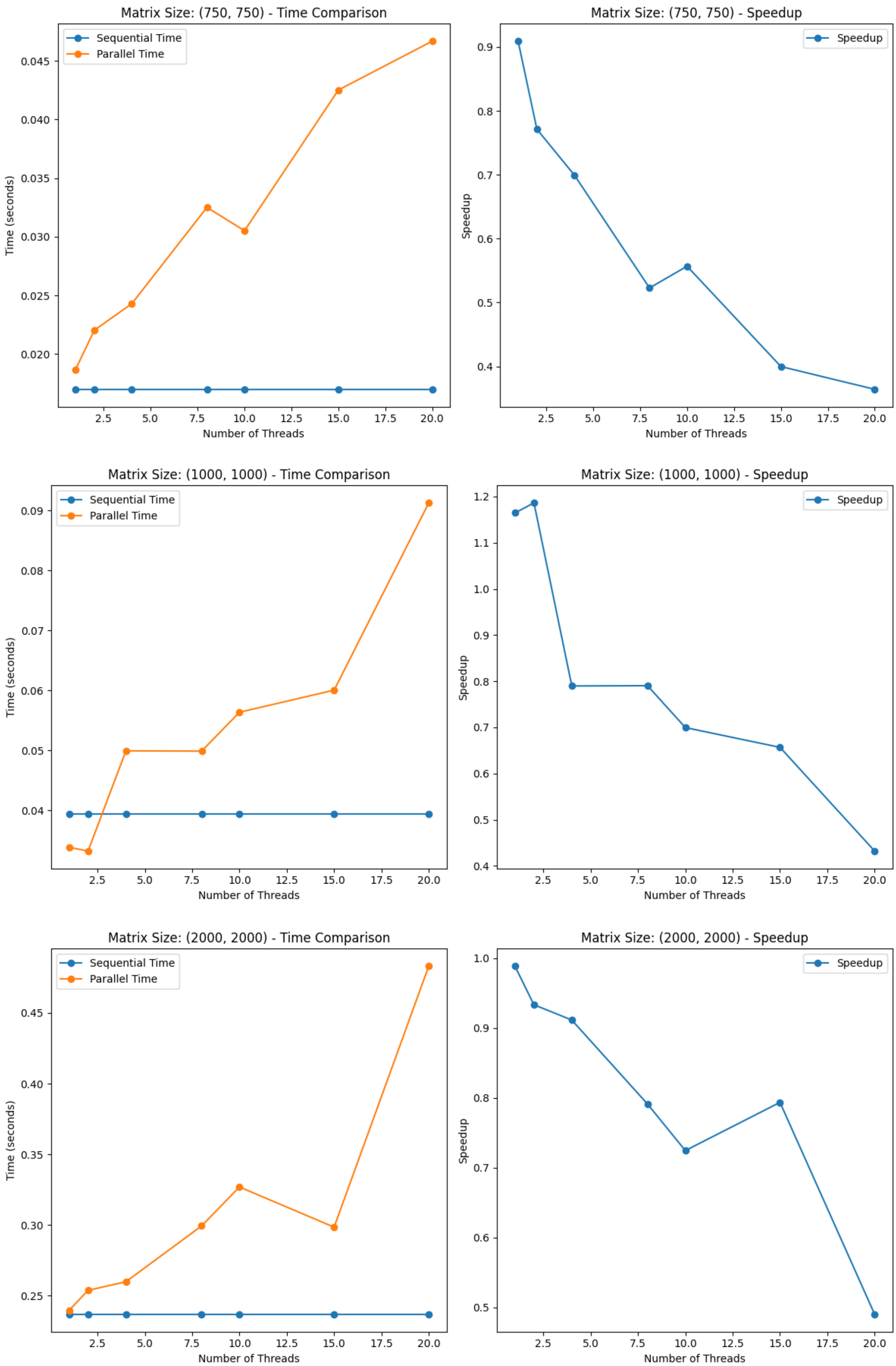
Display a visualization of performance comparison between serial, parallel and Numpy code

visualization of performance comparison between serial, parallel using Numpy code

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

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

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

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

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

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

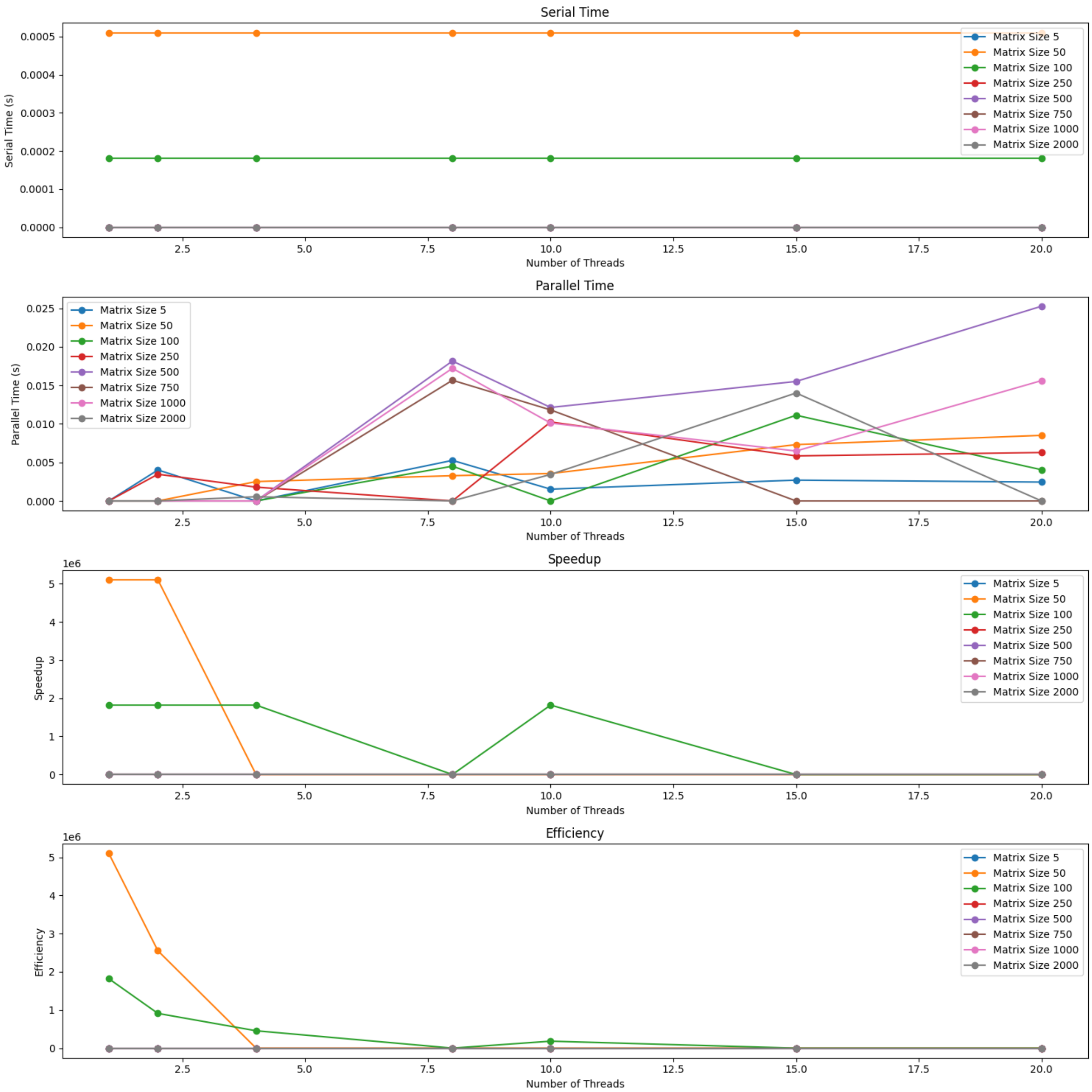



Matrix Size: (100, 100) - Time Comparison
Time (seconds)
Number of Threads
Sequential Time
Parallel Time
Matrix Size: (100, 100) - Speedup
Speedup
Number of Threads
Speedup
Matrix Size: (250, 250) - Time Comparison
Time (seconds)
Number of Threads
Sequential Time
Parallel Time
Matrix Size: (250, 250)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



visualization of performance comparison between serial, parallel using NumPY code

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

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

    plt.subplot(4, 1, 2)
    for size in matrix_sizes:
        data = df2[df2['Matrix Size'] == size]
        plt.plot(data['Threads'], data['Parallel Time'], label=f'Matrix Size {size}', marke
    plt.title('Parallel Time')
    plt.xlabel('Number of Threads')
    plt.ylabel('Parallel Time (s)')
    plt.legend()

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

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

    plt.tight_layout()
    plt.show()


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



Conclusion

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

```
In [ ]:
```