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
```

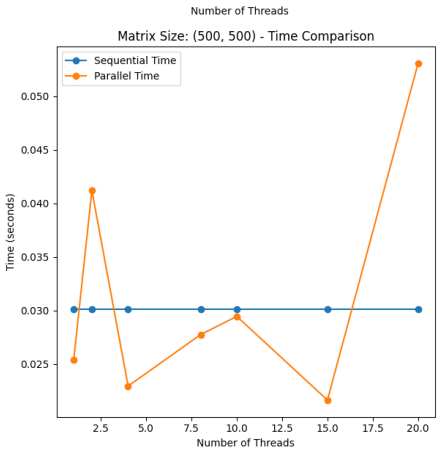
```
def multiply_matrix(A, B, result, start_row, end_row):
    try:
        for i in range(start_row, end_row):
            for j in range(N):
                result[i, j] = 0
                for k in range(N):
                    result[i, j] += A[i, k] * B[k, j]
    except NameError as e:
        pass


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
```

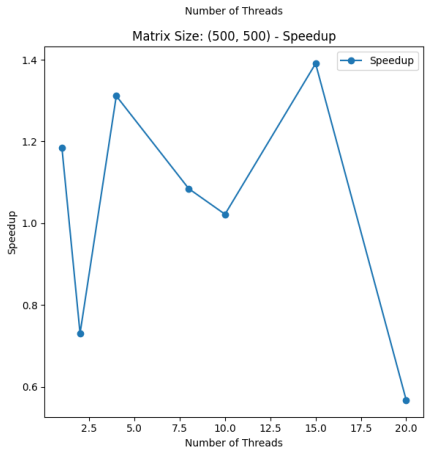
24	250	8	0.000118	0.000706	0.166554	0.020819
25	250	10	0.000118	0.000782	0.150259	0.015026
26	250	15	0.000118	0.001207	0.097373	0.006492
27	250	20	0.000118	0.001580	0.074415	0.003721
28	500	1	0.000134	0.000128	1.046729	1.046729
29	500	2	0.000134	0.000216	0.618102	0.309051
30	500	4	0.000134	0.000359	0.372340	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

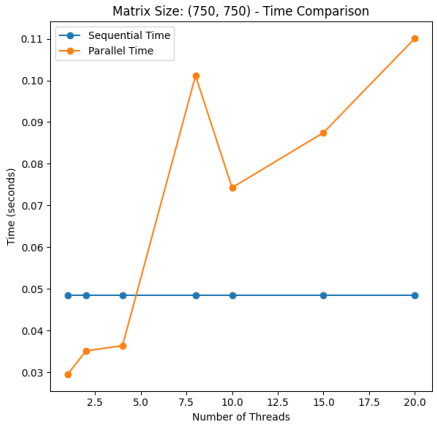
Display a visualization of performance comparison between serial, parallel and NumPY code

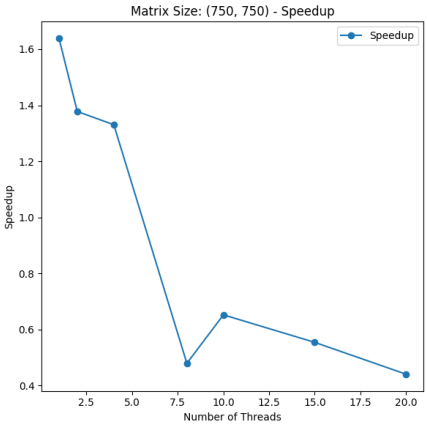
visualization of performance comparison between serial, parallel using NumPY code

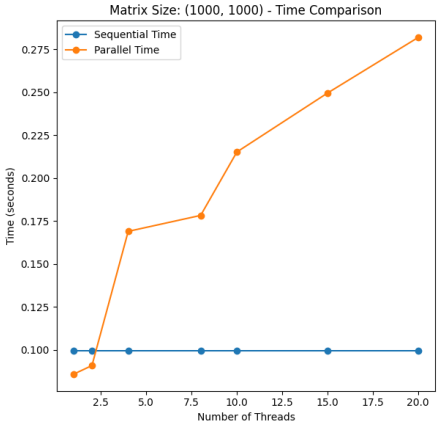
```
import matplotlib.pyplot as plt
matrix_sizes = df['Matrix Size'].unique()

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()
```

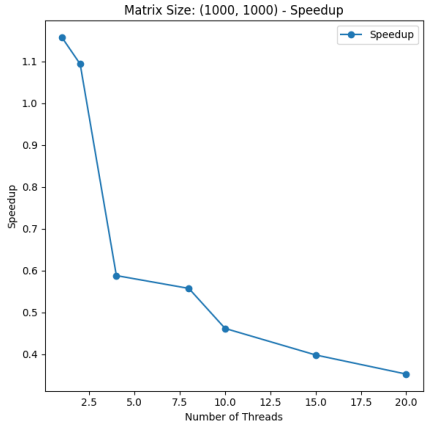












Matrix Size: (2000, 2000) - Time Comparison

Matrix Size: (2000, 2000) - Speedup