МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНОМУ УНІВЕРСИТЕТІ “ЛЬВІВСЬКА ПОЛІТЕХНІКА”
Кафедра систем штучного інтелекту

Лабораторна робота
з дисципліни
« Технології розподілених систем та паралельних обчислень»

Виконав:
студент групи КН-314
Ляшеник О. А
Викладач:
Мочурад Л. І.

Лабораторна робота №1

Тема: Основні конструкції OpenMP. Модель даних

Мета: Ознайомитися з технологією OpenMP та набути практичних навиків її використання.

Хід роботи

Функціональний код (Python) :

```
import numpy as np
import math
import threading
import time
import matplotlib.pyplot as plt

# Функція для обчислення значення елемента матриці
def calculate_matrix_value(i, j):
    return math.sin(i + j)

# Функція для обчислення значення елемента вектора
def calculate_vector_value(i):
    return math.cos(i)

# Функція для заповнення матриці та вектора даними
def fill_data(matrix_size):
    n = matrix_size
    matrix = np.zeros((n, n))
    vector = np.zeros(n)
    for i in range(n):
        for j in range(n):
            matrix[i, j] = calculate_matrix_value(i, j)
        vector[i] = calculate_vector_value(i)
    return matrix, vector

# Функція для секвенційного множення матриці на вектор
```

```python
def multiply_sequential(matrix, vector):
    return np.dot(matrix, vector)

# Робоча функція для паралельного множення, обчислює частину результату

def worker_multiply_threads(matrix, vector, result, start, end):
    """Функція для обчислення частини результату множення матриці на вектор використовуючи потоки"""
    for i in range(start, end):
        result[i] = np.dot(matrix[i], vector)

def multiply_parallel_threads(matrix, vector, num_threads):
    """Функція для паралельного множення використовуючи потоки"""
    n = len(matrix)
    result = np.zeros(n)
    threads = []
    chunk_size = n // num_threads
    for i in range(num_threads):
        start = i * chunk_size
        end = start + chunk_size if i < num_threads - 1 else n
        t = threading.Thread(target=worker_multiply_threads, args=(matrix, vector, result, start, end))
        threads.append(t)
        t.start()

    for t in threads:
        t.join()

    return result
```

```python
if __name__ == "__main__":
    n = 90
    matrix, vector = fill_data(n)

    # Секвенційне множення для порівняння
    start_seq = time.perf_counter()
    result_seq = multiply_sequential(matrix, vector)
    end_seq = time.perf_counter()
    seq_time = end_seq - start_seq
    print(f"Sequential time: {seq_time:.6f} s")

    # Паралельне множення
    num_threads = [1, 2, 4, 8]
    parallel_times = []
    efficiencies = []
    for thread in num_threads:
        start_par = time.perf_counter()
        result_par = multiply_parallel_threads(matrix,
vector, thread)
        end_par = time.perf_counter()
        par_time = end_par - start_par
        parallel_times.append(par_time)

        speedup = seq_time / par_time
        efficiency = (speedup / thread) * 100
        efficiencies.append(efficiency)

        print(f"threads - {thread}")
        print(f"Parallel time: {par_time:.6f} s")
        print(f"Parallel Speedup: {speedup:.2f}, Parallel
Efficiency: {efficiency:.2f}%")
```

```
    # Створюємо графіки
    plt.figure(figsize=(10, 5))

    # Графік часу виконання
    plt.subplot(1, 2, 1)
    plt.plot(num_threads, parallel_times, marker='o',
linestyle='-', color='b')
    plt.plot(num_threads, [seq_time] * len(num_threads),
linestyle='--', color='r', label='Sequential')
    plt.title('Parallel Execution Time')
    plt.xlabel('Number of Threads')
    plt.ylabel('Time (seconds)')
    plt.legend()

    # Графік ефективності
    plt.subplot(1, 2, 2)
    plt.plot(num_threads, efficiencies, marker='o',
linestyle='-', color='g')
    plt.title('Parallel Efficiency')
    plt.xlabel('Number of Threads')
    plt.ylabel('Efficiency (%)')

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

Результати виводу:

Sequential time: 0.000025 s

threads - 1

Parallel time: 0.000941 s

Parallel Speedup: 0.03, Parallel Efficiency: 2.61%

threads - 2

Parallel time: 0.000803 s

Parallel Speedup: 0.03, Parallel Efficiency: 1.53%

threads - 4

Parallel time: 0.001266 s

Parallel Speedup: 0.02, Parallel Efficiency: 0.49%

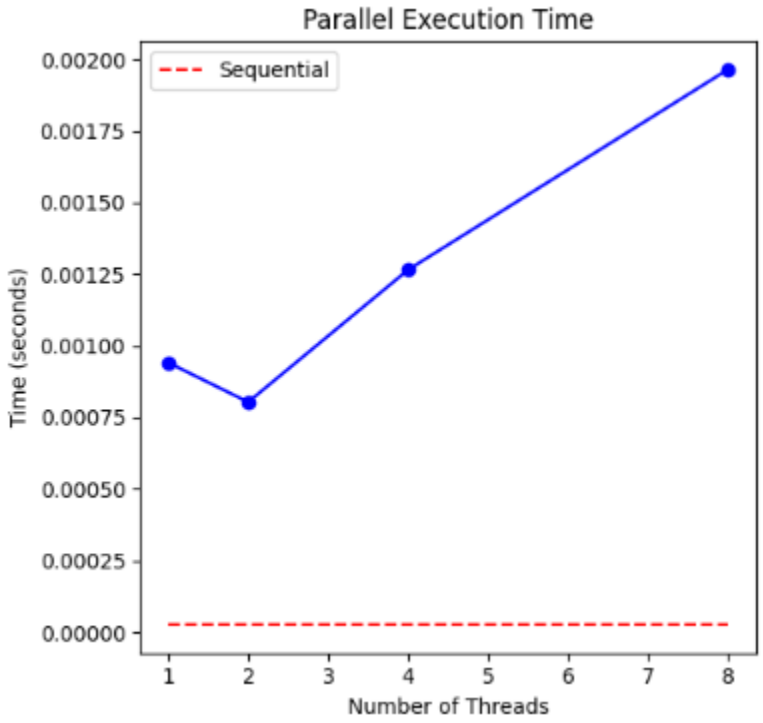
threads - 8

Parallel time: 0.001965 s

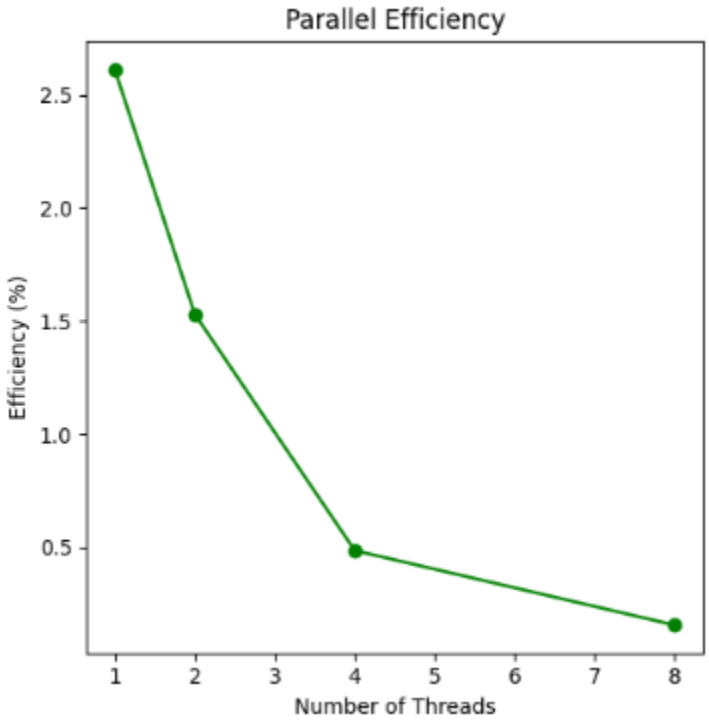
Parallel Speedup: 0.01, Parallel Efficiency: 0.16%

Графіки:

Графік паралельного прискорення :



Графік паралельної ефективності:



Зваэаючи на первну випадковість ефективності пристрою при виконанні програми, потрібно запустити програму ще декілька раз для більш точної інформації

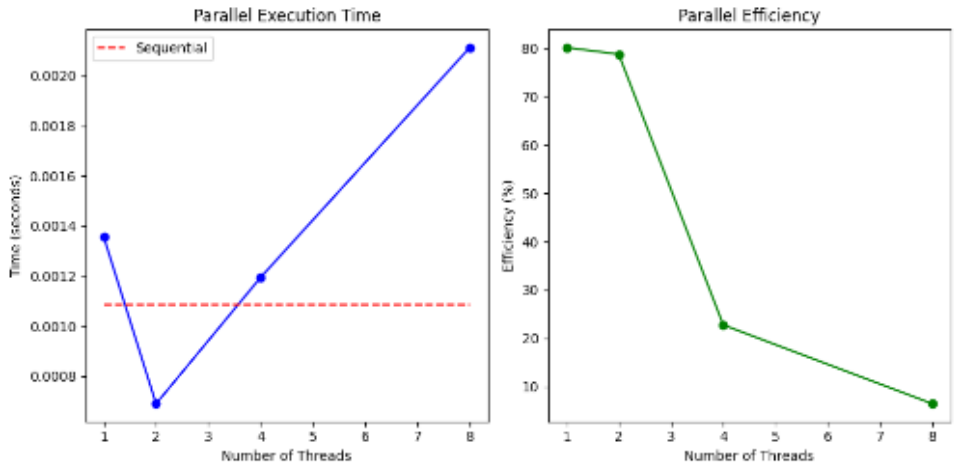
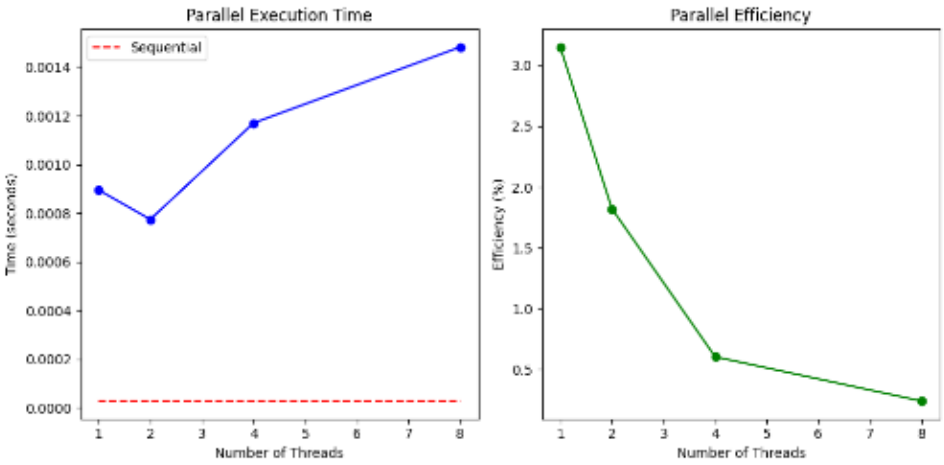


Рис.1.3



Коментар : Бачимо, що при збільшенній кількості потоків - збільшується і час на виконання, це спричинено тим, що програма є занадто швидко у виконанні і саме створення потоків займає більше часу. Не зважаючи на це, найбільшу ефективність показало саме ввиконання, з двома потоками.

Висновок:

Я ознайомився з технологією розподіленого виконання, та дізнався як її застосовувати.