**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  
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  
  
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%

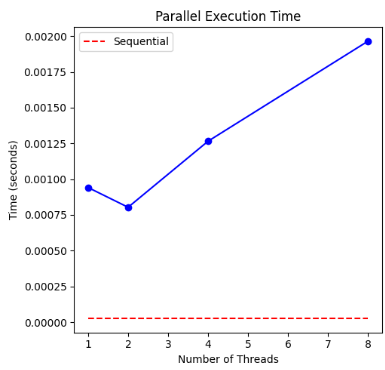
threads - 8

Parallel time: 0.001965 s

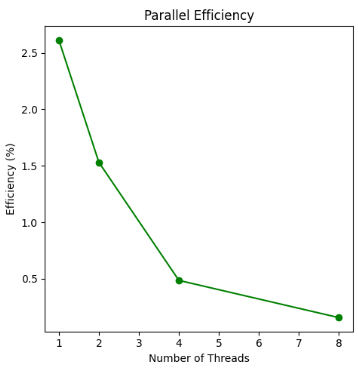
Parallel Speedup: 0.01, Parallel Efficiency: 0.16%

**Графіки:**

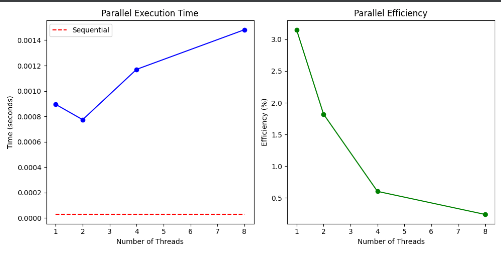
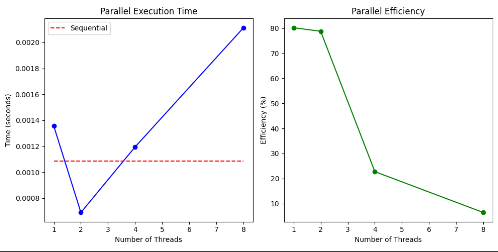
Графік паралельного прискорення :



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.