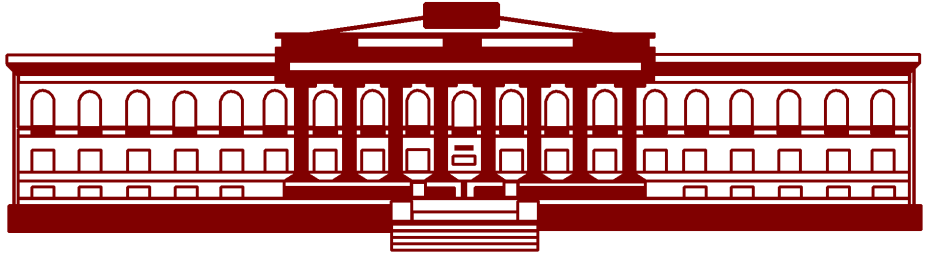
**КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ**

**імені ТАРАСА ШЕВЧЕНКА**

Факультет інформаційних технологій

**Кафедра прикладних інформаційних систем**

****

**Звіт**

до лабораторної роботи №2

з дисципліни «**Паралельні та розподілені обчислення**»

ВАРІАНТ №16

**Виконано:**

студ. групи ПП- 31, підгрупа 2

Селецький Віктор

**Київ – 2023**

Тема: Створення програми, яка використовує паралельні потоки ОС

Ціль: освоїти реалізацію паралельних обчислень за допомогою

стандартних засобів багатопоточності сучасних ОС та виконати

порівняльне оцінювання часу виконання програми з послідовним обчисленням.

Завдання: На базі однієї із операційних систем ПК (за вибором студента) та на

двох з мов програмування (С++ / Java (не використовувати java.util.concurrent ) /

Python /тощо) створити програмний застосунок з кількома потоками

(threads) однієї програми (процесу), який підраховує суму перших N членів

заданої послідовності (формулу для підрахунку i-го елемента взяти з таблиці

1).

Програма повинна виконувати такі дії:

- запитувати значення змінної N

- у циклі від 0 до N розраховувати значення i-го елемента послідовності

ai. Додавати це значення до суми. Виводити результат суми та

витраченого на розрахунок часу на екран.

Підібрати таке значення N, щоб програма виконувалася кілька десятків

секунд (50-500 с). Для цього почати з маленьких значень, поступово

збільшуючи їх.

У звіті навести код програми, результати її виконання: суму та час

виконання за кількох значень N.

Зробити загальну таблицю порівняння результатів для обох мов

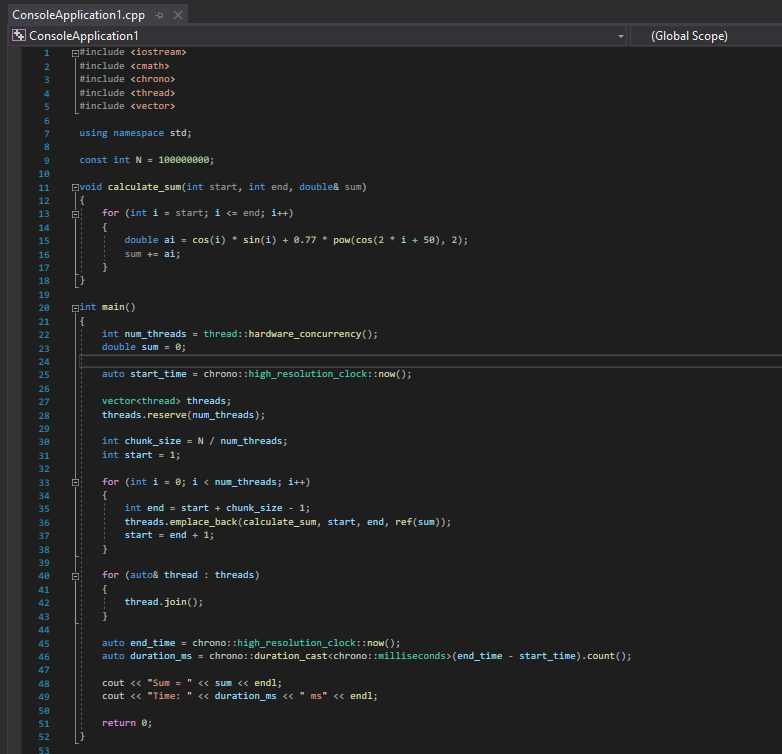
програмування, зробити висновки. Побудувати графіки залежностей часу

виконання від N для обох програм. Порівняти результати з результатами

практичної роботи No1.



Виконання:



Код:

#include <iostream>

#include <cmath>

#include <chrono>

#include <thread>

#include <vector>

using namespace std;

const int N = 100000000;

void calculate\_sum(int start, int end, double& sum)

{

for (int i = start; i <= end; i++)

{

double ai = cos(i) \* sin(i) + 0.77 \* pow(cos(2 \* i + 50), 2);

sum += ai;

}

}

int main()

{

int num\_threads = thread::hardware\_concurrency();

double sum = 0;

auto start\_time = chrono::high\_resolution\_clock::now();

vector<thread> threads;

threads.reserve(num\_threads);

int chunk\_size = N / num\_threads;

int start = 1;

for (int i = 0; i < num\_threads; i++)

{

int end = start + chunk\_size - 1;

threads.emplace\_back(calculate\_sum, start, end, ref(sum));

start = end + 1;

}

for (auto& thread : threads)

{

thread.join();

}

auto end\_time = chrono::high\_resolution\_clock::now();

auto duration\_ms = chrono::duration\_cast<chrono::milliseconds>(end\_time - start\_time).count();

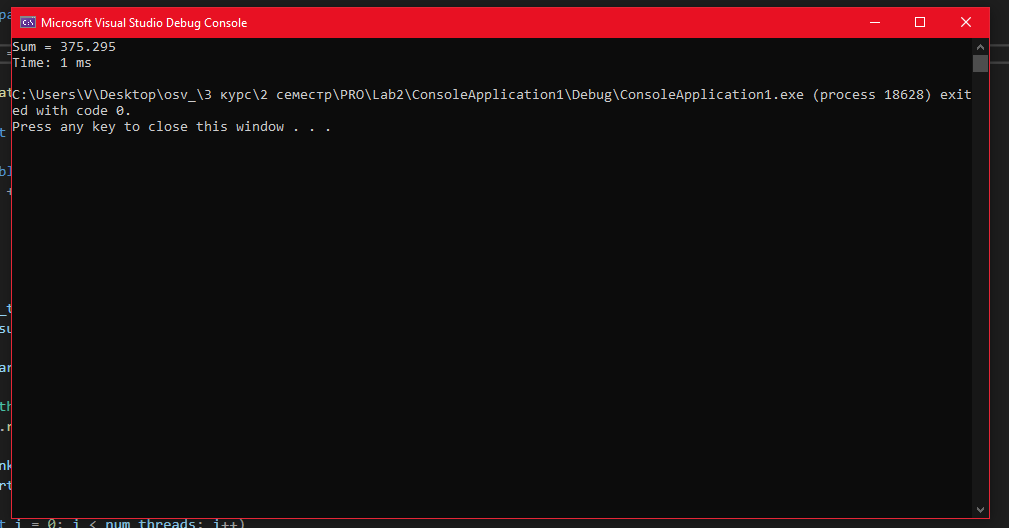
cout << "Sum = " << sum << endl;

cout << "Time: " << duration\_ms << " ms" << endl;

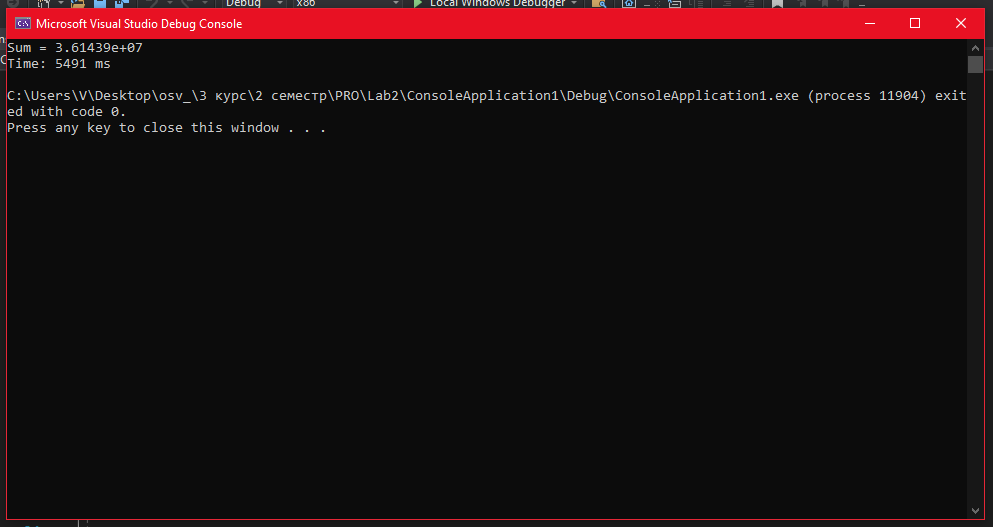
return 0;

}

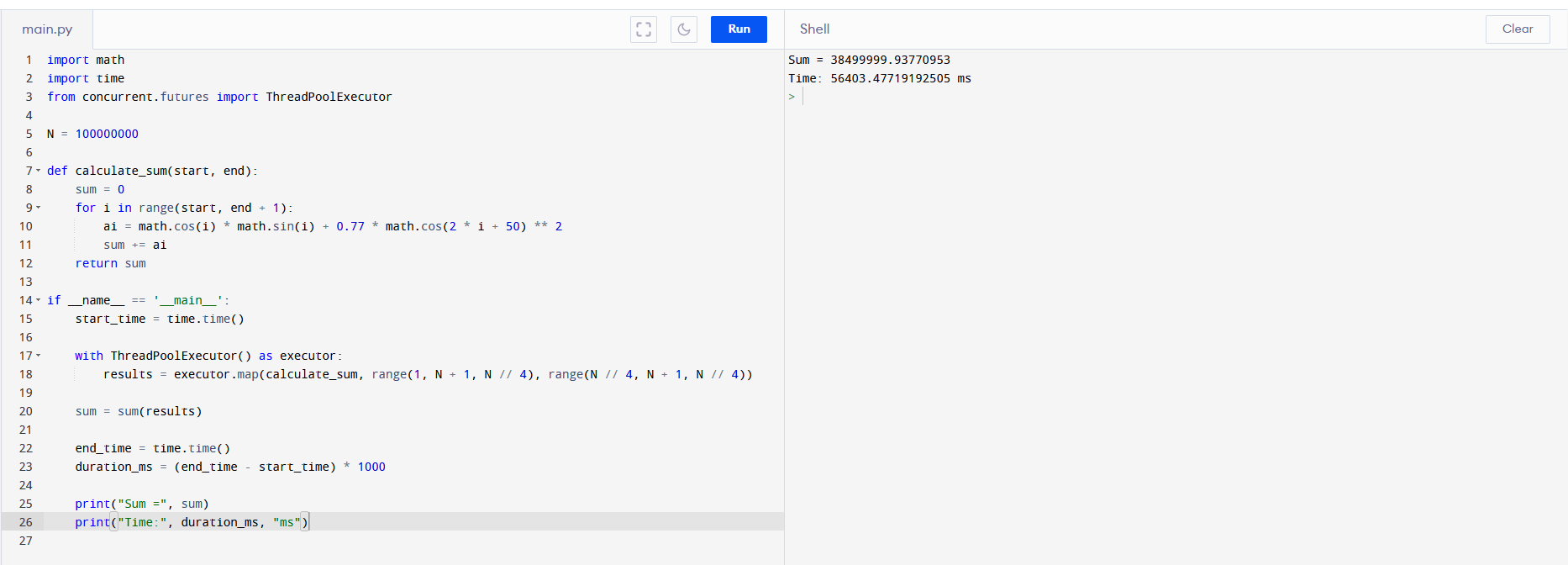
Результат виконання:



(1000 членів прогресії)



(100000000 членів прогресії)



N = 1 000 000 00

Код:

import math

import time

from concurrent.futures import ThreadPoolExecutor

N = 100000000

def calculate\_sum(start, end):

sum = 0

for i in range(start, end + 1):

ai = math.cos(i) \* math.sin(i) + 0.77 \* math.cos(2 \* i + 50) \*\* 2

sum += ai

return sum

if \_\_name\_\_ == '\_\_main\_\_':

start\_time = time.time()

with ThreadPoolExecutor() as executor:

results = executor.map(calculate\_sum, range(1, N + 1, N // 4), range(N // 4, N + 1, N // 4))

sum = sum(results)

end\_time = time.time()

duration\_ms = (end\_time - start\_time) \* 1000

print("Sum =", sum)

print("Time:", duration\_ms, "ms")

Таблиця для порівняння:

| **Кількість членів** | **C++ (без мультипоточності)** | **C++ (з мультипоточністю)** | **Python (без мультипоточності)** | **Python (з мультипоточністю)** |
| --- | --- | --- | --- | --- |
| 100 | 0.000008 сек | 0.000033 сек | 0.001820 сек | 0.000340 сек |
| 1000 | 0.000033 сек | 0.000048 сек | 0.000042 сек | 0.000076 сек |
| 10000 | 0.000310 сек | 0.000218 сек | 0.000571 сек | 0.000295 сек |
| 999999 | 0.025255 сек | 0.004491 сек | 0.056196 сек | 0.013505 сек |