

**МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ  
 НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ  
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ   
імені ІГОРЯ СІКОРСЬКОГО»**

**Навчально-науковий інститут прикладного системного аналізу  
Кафедра системного проектування**

**Звіт**

**про виконання практичної роботи №1  
з дисципліни «Паралельні обчислення»**

Виконав:  
студент III курсу, групи ДА-22  
Моренець Сергій Андрійович

Прийняв:

асистент Яременко В. С.

Київ – 2025

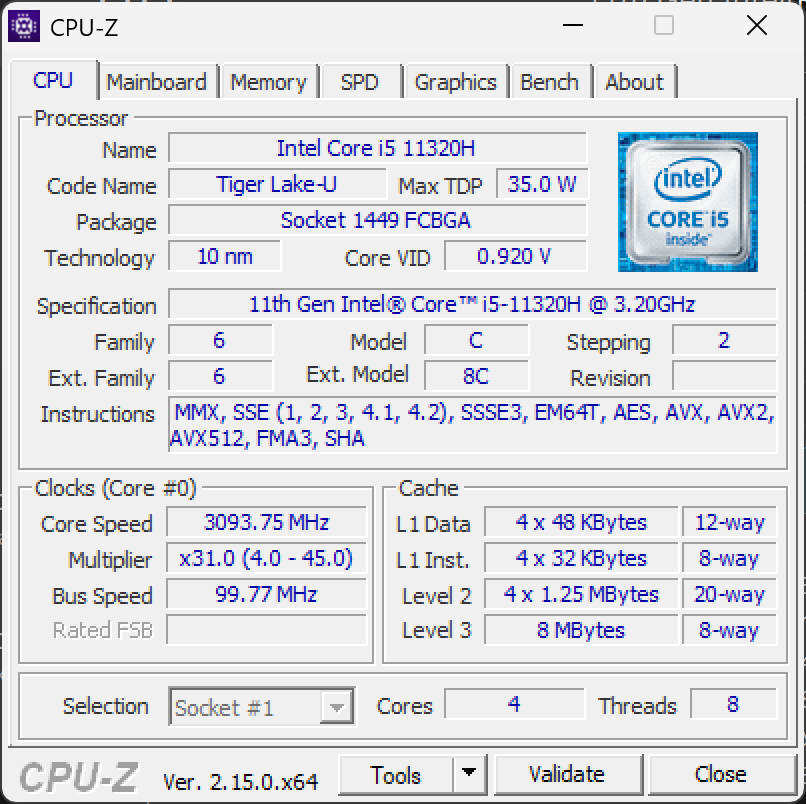
***Завдання:***

1. Визначити основні характеристики ПК, котрі на думку студента впливають на ефективність виконання паралельних обчислень. Зафіксувати значення даних характеристик для ПК студента, та занести їх до протоколу роботи.
2. Створити або використати наявній механізм для заміру часу виконання програми, або інших параметрів, котрі студент вважає релевантними. Занести опис механізму до роботи.
3. Вирішити обрану за варіантом задачу, не використовуючи паралелізацію. Заміряти час вирішення задачі, або інші параметри, котрі студент вважає релевантними.
4. Вирішити оборану за варіантом задачу, використовуючи паралелізацію. Заміряти час вирішення задачі, або інші параметри, котрі студент вважає релевантними. Обґрунтувати вибір алгоритму паралелізації (розподілення даних між потоками), надати опис та обґрунтування в протоколі роботи.
5. Перевірити алгоритм на фіксованих кількостях потоків: 2-рази меншій, ніж кількість фізичних ядер, на кількості рівній фізичним ядрам, на кількості рівній логічних ядрам, на кількості більшій в 2, 4, 8, 16 разів ніж кількість логічних ядер.
6. Повторити пункт 5 з використанням різної розмірності даних.
7. Зробити таблиці та графіки залежності часу виконання завдання від кількості потоків для різної розмірності даних. Надати опис графіків, з причинами виникнення отриманих результатів в протоколі роботи.
8. Надати висновок, що повинен містити аналіз отриманих результатів.

***Варіант 20:***

Заповнити квадратну матрицю випадковими числами. На побічній діагоналі розмістити мінімальний елемент рядка.

Загальна інформація про процесор на основі даних з CPU-Z:



Кількість фізичних ядер – 4

Кількість логічних ядер – 8

Створимо програму, що виконує завдання за варіантом примітивним алгоритмом та алгоритмом з паралелізацією з різною кількістю потоків. Порівняємо час виконання завдання.

#include <iostream>

#include <chrono>

#include <thread>

#include <cstdlib>

#include <ctime>

#include <vector>

#include <random>

#define PHYSICAL\_CORES 4

#define LOGICAL\_CORES 8

using namespace std;

using namespace std::chrono;

void simpleAlgorithm(int matrixDimensions) {

    int minElemIndex;

    vector<vector<int>> matrix(matrixDimensions, vector<int>(matrixDimensions));

    random\_device rd;

    mt19937 gen(rd());

    uniform\_int\_distribution<> distrib(0, 99);

    for (int i = 0; i < matrixDimensions; i++) {

        for (int j = 0; j < matrixDimensions; j++) {

            matrix[i][j] = distrib(gen);

        }

    }

    for (int i = 0; i < matrixDimensions; i++) {

        minElemIndex = 0;

        for (int j = 1; j < matrixDimensions; j++) {

            if (matrix[i][j] < matrix[i][minElemIndex]) {

                minElemIndex = j;

            }

        }

        swap(matrix[i][minElemIndex], matrix[i][matrixDimensions - i - 1]);

    }

}

void fillRandomValue(vector<vector<int>>& matrix, int startRow, int endRow) {

    random\_device rd;

    mt19937 gen(rd());

    uniform\_int\_distribution<> distrib(0, 99);

    int matrixDimensions = matrix.size();

    for (int i = startRow; i < endRow; ++i) {

        for (int j = 0; j < matrixDimensions; ++j) {

            matrix[i][j] = distrib(gen);

        }

    }

}

void processRows(vector<vector<int>>& matrix, int startRow, int endRow) {

    int matrixDimensions = matrix.size();

    for (int i = startRow; i < endRow; ++i) {

        int minElemIndex = 0;

        for (int j = 1; j < matrixDimensions; ++j) {

            if (matrix[i][j] < matrix[i][minElemIndex]) {

                minElemIndex = j;

            }

        }

        swap(matrix[i][minElemIndex], matrix[i][matrixDimensions - i - 1]);

    }

}

void parallelizationAlgorithm (int matrixDimensions, int threadsCount) {

    vector<vector<int>> matrix(matrixDimensions, vector<int>(matrixDimensions));

    vector<thread> threads;

    int rowsPerThread = matrixDimensions / threadsCount;

    int remainingRows = matrixDimensions % threadsCount;

    int startRow = 0;

    for (int i = 0; i < threadsCount; ++i) {

        int endRow = startRow + rowsPerThread;

        if (i < remainingRows) {

            endRow++;

        }

        threads.emplace\_back(fillRandomValue, ref(matrix), startRow, endRow);

        startRow = endRow;

    }

    for (int i = 0; i < threads.size(); ++i) {

        threads[i].join();

    }

    threads.clear();

    startRow = 0;

    for (int i = 0; i < threadsCount; ++i) {

        int endRow = startRow + rowsPerThread;

        if (i < remainingRows) {

            endRow++;

        }

        threads.emplace\_back(processRows, ref(matrix), startRow, endRow);

        startRow = endRow;

    }

    for (int i = 0; i < threads.size(); ++i) {

        threads[i].join();

    }

}

void results() {

    vector<int> matrixDimensions= {1000, 5000, 10000, 20000};

    vector<int> threadsCount = {

        PHYSICAL\_CORES / 2,

        PHYSICAL\_CORES,

        LOGICAL\_CORES,

        LOGICAL\_CORES \* 2,

        LOGICAL\_CORES \* 4,

        LOGICAL\_CORES \* 8,

        LOGICAL\_CORES \* 16

    };

    cout << "Size\t|Threads | Time" << endl;

    cout << "------------------------" << endl;

    for (int i = 0; i < matrixDimensions.size(); i++) {

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

        simpleAlgorithm(matrixDimensions[i]);

        auto stop = high\_resolution\_clock::now();

        auto duration = duration\_cast<milliseconds>(stop - start);

        cout << matrixDimensions[i] << "\t| " << 1 << "\t |" << duration.count() << " ms" << endl;

        for (int j = 0; j < threadsCount.size(); j++) {

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

            parallelizationAlgorithm(matrixDimensions[i], threadsCount[j]);

            auto stop = high\_resolution\_clock::now();

            auto duration = duration\_cast<milliseconds>(stop - start);

            cout << matrixDimensions[i] << "\t| " << threadsCount[j] << "\t |" << duration.count() << " ms" << endl;

        }

        cout << "- - - - - - - - - - - - " << endl;

    }

}

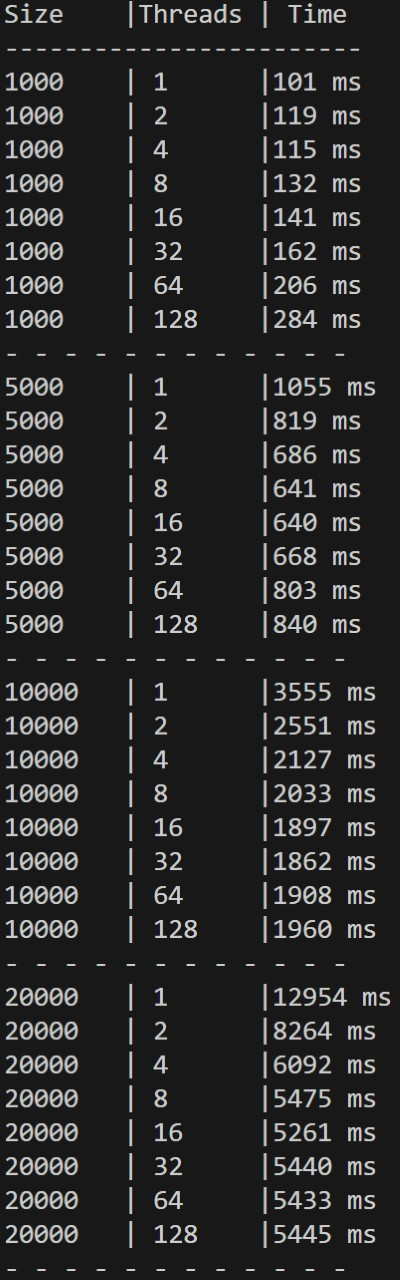
int main() {

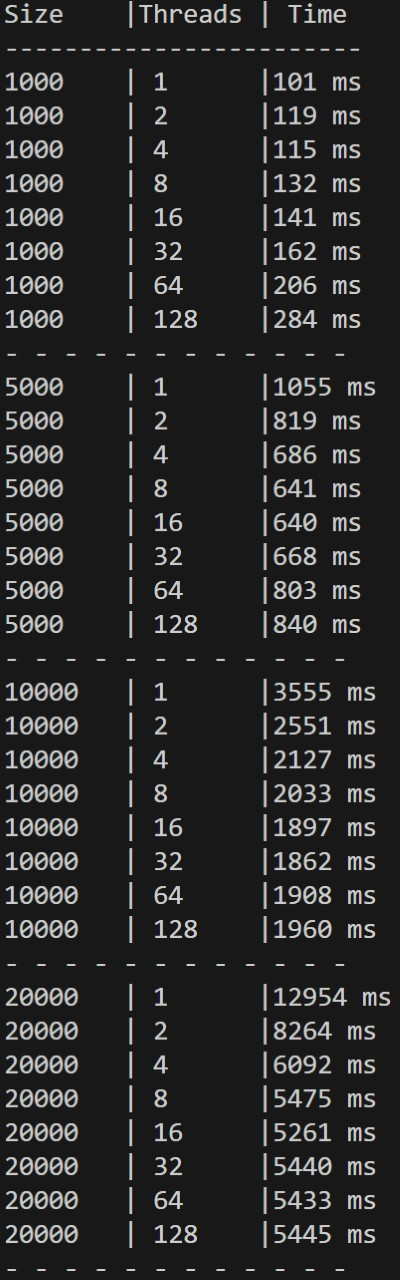
    results();

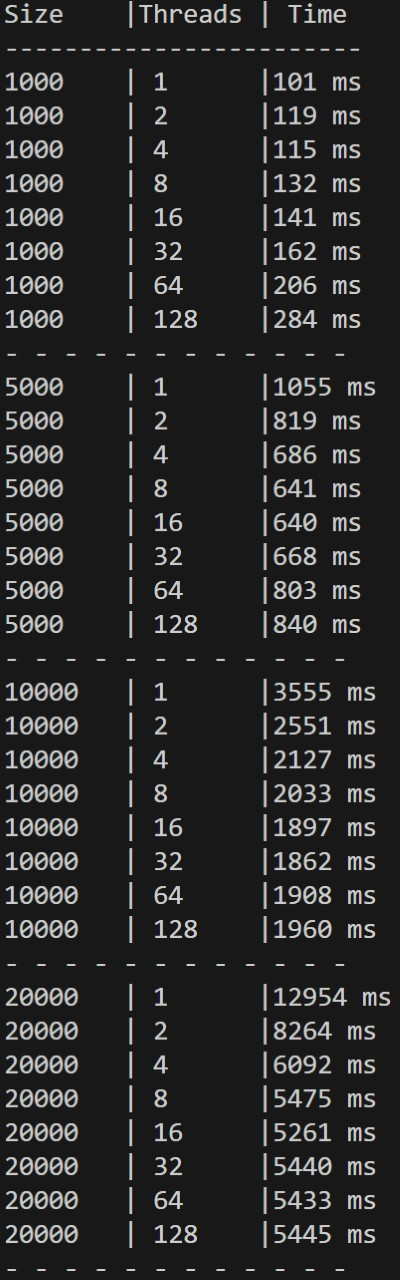
    return 0;

}

Результат виконання:







Представимо отримані результати у вигляді графіків залежності часу від кількості потоків. (1 потік – виконання примітивним алгоритмом).

***Висновки:***

У ході лабораторної роботи було досліджено ефективність паралельної обробки матриць різних розмірів. Для невеликих матриць використання багатьох потоків лише збільшувало час виконання через накладні витрати. При більших розмірах паралелізація дозволила значно скоротити час обробки, особливо при використанні 8–32 потоків. Збільшення кількості потоків понад 32 не давало суттєвого покращення і навіть інколи погіршувало результат через перевантаження системи. Загалом, паралельна обробка ефективна для великих задач, але потребує правильного вибору кількості потоків залежно від розміру задачі та ресурсів системи.

Git: <https://github.com/m0renets/PC_Lab_1.git>