

**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  
Жадько Микита Сергійович

Прийняв:

асистент Яременко В. С.

Київ – 2025

**ПОСТАНОВКА ЗАДАЧІ**

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.

Завдання за варіантом (№9)

Заповнити квадратну матрицю випадковими числами. На головній діагоналі розмістити суми елементів, які лежать в тому ж рядку.

**ХІД РОБОТИ**

Визначимо основні характеристики за допомогою програми AIDA64 Extreme.



Рисунок 1 — Загальна інформація про комп’ютер.

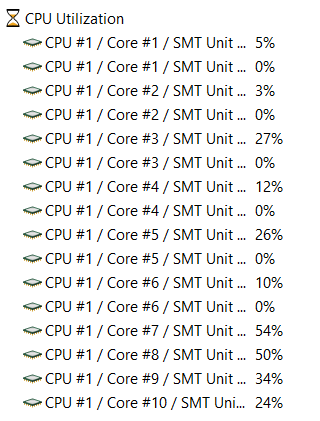


Рисунок 2 — Завантаженість 16 ядер у випадковий момент часу.

Вирішимо всі завдання однією програмою, яка виконуватиме лінійні та паралельні обчислення на масивах різної величини.

Так як завдання полягає у знаходженні суми елементів рядка, можна організувати роботу потоків навіть без інструментів контролю (на кшталт м’ютексів та інших). Треба просто різним потокам призначити свій діапазон строк, суму елементів яких їм треба знайти.

Код програми:

#include <iostream>

#include <chrono>

#include <thread>

#include <vector>

#include <windows.h>

#include <intrin.h>

#include <iomanip>

#define SeedNum 7

#define ShouldCheckCorrectness 1

**using** **namespace** std**;**

**using** chrono**::**nanoseconds**;**

**using** chrono**::**duration\_cast**;**

**using** chrono**::**high\_resolution\_clock**;**

void printCacheInfo**()** **{**

int CPUInfo**[**4**];**

\_\_cpuid**(**CPUInfo**,** 0**);**

int nIds **=** CPUInfo**[**0**];**

**for** **(**int i **=** 0**;** i **<=** nIds**;** **++**i**)** **{**

\_\_cpuidex**(**CPUInfo**,** i**,** 0**);**

**if** **(**CPUInfo**[**0**]** **==** 4**)** **{**

int cacheLevel **=** **(**CPUInfo**[**0**]** **>>** 5**)** **&** 0x7**;**

int cacheType **=** CPUInfo**[**0**]** **&** 0xFF**;**

**if** **(**cacheType **==** 1 **||** cacheType **==** 2 **||** cacheType **==** 3**)** **{**

cout **<<** "Cache Level: " **<<** cacheLevel **<<** ", Type: "

**<<** **(**cacheType **==** 1 **?** "Data" **:** **(**cacheType **==** 2 **?** "Instruction" **:** "Unified"**))**

**<<** ", Size: " **<<** **((**CPUInfo**[**1**]** **+** 1**)** **\*** 8**)** **<<** " KB" **<<** endl**;**

**}**

**}**

**}**

**}**

void printSystemInfo**(**int **&**cpuNum**)** **{**

SYSTEM\_INFO sysinfo**;**

GetSystemInfo**(&**sysinfo**);**

cpuNum **=** sysinfo**.**dwNumberOfProcessors**;**

cout **<<** "System Information:" **<<** endl**;**

cout **<<** "Number of logical processors: " **<<** sysinfo**.**dwNumberOfProcessors **<<** endl**;**

cout **<<** "Processor architecture: "

**<<** **(**sysinfo**.**wProcessorArchitecture **==** PROCESSOR\_ARCHITECTURE\_INTEL **?** "x86" **:**

sysinfo**.**wProcessorArchitecture **==** PROCESSOR\_ARCHITECTURE\_AMD64 **?** "x64" **:** "Unknown"**)**

**<<** endl**;**

cout **<<** "Page size: " **<<** sysinfo**.**dwPageSize **<<** " bytes" **<<** endl**;**

cout **<<** "Minimum application address: " **<<** sysinfo**.**lpMinimumApplicationAddress **<<** endl**;**

cout **<<** "Maximum application address: " **<<** sysinfo**.**lpMaximumApplicationAddress **<<** endl**;**

MEMORYSTATUSEX memInfo**;**

memInfo**.**dwLength **=** **sizeof(**MEMORYSTATUSEX**);**

GlobalMemoryStatusEx**(&**memInfo**);**

cout **<<** "Total physical memory: " **<<** memInfo**.**ullTotalPhys **/** **(**1024 **\*** 1024**)** **<<** " MB" **<<** endl**;**

cout **<<** "Available physical memory: " **<<** memInfo**.**ullAvailPhys **/** **(**1024 **\*** 1024**)** **<<** " MB" **<<** endl**;**

printCacheInfo**();**

**}**

void processMatrixSection**(**int startRow**,** int endRow**,** const vector**<**vector**<**int**>>&** primaryMatrix**,** vector**<**vector**<**int**>>&** matrix**)** **{**

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

int rowSum **=** 0**;**

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

rowSum **+=** primaryMatrix**[**i**][**j**];**

**}**

matrix**[**i**][**i**]** **=** rowSum**;**

**}**

**}**

bool checkMatrixCorrectness**(**const vector**<**vector**<**int**>>&** matrix**,** const vector**<**vector**<**int**>>&** primaryMatrix**,** int randomRowCount **=** 10**)** **{**

vector**<**int**>** randomRows**;**

**for** **(**int i **=** 0**;** i **<** randomRowCount**;** **++**i**)** **{**

randomRows**.**push\_back**(**rand**()** **%** matrix**.**size**());**

**}**

bool isCorrect **=** **true;**

**for** **(**auto row **:** randomRows**)** **{**

**if** **(**row **>=** matrix**.**size**())** **continue;**

int actualSum **=** 0**;**

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

actualSum **+=** primaryMatrix**[**row**][**j**];**

**}**

**if** **(**matrix**[**row**][**row**]** **!=** actualSum**)** **{**

cout **<<** "Error in row " **<<** row **<<** ": Expected " **<<** actualSum **<<** ", but got " **<<** matrix**[**row**][**row**]** **<<** endl**;**

isCorrect **=** **false;**

**}**

**}**

**return** isCorrect**;**

**}**

void linearProcessMatrix**(**vector**<**vector**<**int**>>&** matrix**)** **{**

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

int rowSum **=** 0**;**

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

rowSum **+=** matrix**[**i**][**j**];**

**}**

matrix**[**i**][**i**]** **=** rowSum**;**

**}**

**}**

int main**()** **{**

int cpuNum**;**

printSystemInfo**(**cpuNum**);**

vector matrixSizes **=** **{**

100**,**

1000**,**

5000**,**

20000**,**

50000**,**

**};**

vector numCPUArr **=** **{**

cpuNum **/** 2**,**

cpuNum**,**

cpuNum **\*** 2**,**

cpuNum **\*** 4**,**

cpuNum **\*** 8**,**

cpuNum **\*** 16**,**

**};**

cout **<<** "\nTest Results:" **<<** endl**;**

cout **<<** "Matrix Size\tThreads\tTime (seconds)\tCorrect?" **<<** endl**;**

**for** **(**int matrixSize **:** matrixSizes**)** **{**

vector primaryMatrix**(**matrixSize**,** vector**<**int**>(**matrixSize**));**

srand**(**SeedNum**);**

**for** **(**int i **=** 0**;** i **<** matrixSize**;** **++**i**)** **{**

**for** **(**int j **=** 0**;** j **<** matrixSize**;** **++**j**)** **{**

primaryMatrix**[**i**][**j**]** **=** rand**()** **%** 10001**;**

**}**

**}**

**{**

vector**<**vector**<**int**>>** copiedMatrix **=** primaryMatrix**;**

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

linearProcessMatrix**(**copiedMatrix**);**

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

string correctness **=** ShouldCheckCorrectness **?** **(**checkMatrixCorrectness**(**copiedMatrix**,** primaryMatrix**)** **?** "Yes" **:** "No"**)** **:** "Unknown"**;**

auto elapsed **=** duration\_cast**<**nanoseconds**>(**end **-** start**).**count**()** **\*** 1e-9**;**

cout **<<** endl **<<** matrixSize **<<** "\t\tLinear\t" **<<** fixed **<<** setprecision**(**6**)** **<<** elapsed **<<** "\t" **<<** correctness **<<** endl**;**

**}**

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

vector**<**vector**<**int**>>** copiedMatrix **=** primaryMatrix**;**

int threadsCount **=** numCPUArr**[**i**];**

vector**<**thread**>** threads**;**

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

int rowsPerThread **=** matrixSize **/** threadsCount**;**

int extraRows **=** matrixSize **%** threadsCount**;**

**for** **(**int t **=** 0**;** t **<** threadsCount**;** **++**t**)** **{**

int startRow **=** t **\*** rowsPerThread **+** min**(**t**,** extraRows**);**

int endRow **=** startRow **+** rowsPerThread **+** **(**t **<** extraRows **?** 1 **:** 0**);**

threads**.**emplace\_back**(**processMatrixSection**,** startRow**,** endRow**,** ref**(**primaryMatrix**),** ref**(**copiedMatrix**));**

**}**

**for** **(**auto **&**th **:** threads**)** **{**

**if** **(**th**.**joinable**())** **{**

th**.**join**();**

**}**

**}**

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

string correctness **=** ShouldCheckCorrectness **?** **(**checkMatrixCorrectness**(**copiedMatrix**,** primaryMatrix**)** **?** "Yes" **:** "No"**)** **:** "Unknown"**;**

auto elapsed **=** duration\_cast**<**nanoseconds**>(**end **-** start**).**count**()** **\*** 1e-9**;**

cout **<<** matrixSize **<<** "\t\t" **<<** threadsCount **<<** "\t" **<<** fixed **<<** setprecision**(**6**)** **<<** elapsed **<<** "\t" **<<** correctness **<<** endl**;**

**}**

**}**

**return** 0**;**

**}**

Запустимо програму та занесемо результати роботи в окремі таблиці.

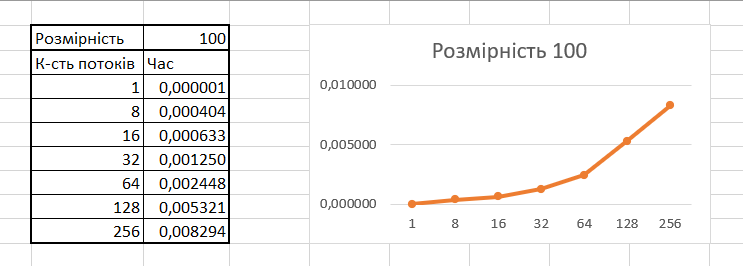


Рисунок 3 — Час роботи потоків з матрицею 100х100.



Рисунок 4 — Час роботи потоків з матрицею 1000х1000.

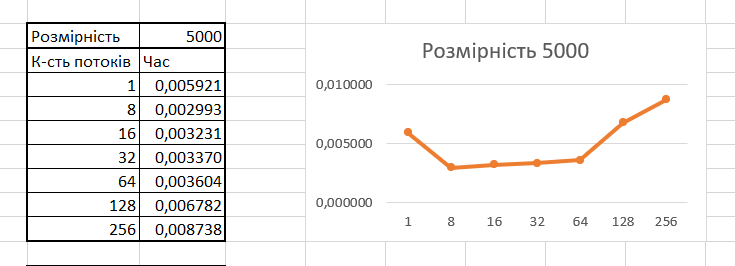


Рисунок 5 — Час роботи потоків з матрицею 5000х5000.



Рисунок 6 — Час роботи потоків з матрицею 20000х20000.



Рисунок 7 — Час роботи потоків з матрицею 50000х50000.

**ВИСНОВКИ**

Було виконано лабораторну роботу, у якій створено програму, що вирішує задану задачу різною кількістю потоків з матрицями різної розмірності. Результати вийшли доволі цікавими.

Якщо поглянути на результат роботи з матрицями розмірності 100 та 1000, можна побачити, що чим менша кількість потоків, тим більш швидким є виконання задачі. Так сталося через те, що на створення потоків, їх організацію та завершення роботи необхідний час та ресурси, які значно перевищують зекономлені ресурси у ході паралельних обчислень.

Розмірність 5000 представляє собою ідеальний варіант розвитку подій: лінійне виконання програми занадто довге для такої розмірності, а якщо збільшувати кількість потоків понад кількість фізичних ядер, це тільки сповільнить виконання програми.

Щодо розмірностей 20000 та 50000, все дуже неоднозначно, програма веде себе так, як не повинна вести. Виконання схоже на роботу із розмірністю 5000, проте більша кількість потоків не означає більший час виконання. Скоріш за все, це відбувається через паралельні процеси, що протікають разом з виконанням програми (робота браузера, тощо).

Також [посилання на репозиторій](https://github.com/nikk0308/Parallel_Calculations).