МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ імені ІВАНА ФРАНКА

**Звіт із**

**Розпаралелення додавання/віднімання матриць**

**Лабораторна робота №1**

з курсу «Паралельні та розподілені обчислення»

Виконав:

Готюк Максим

Група Пмі-33с

Оцінка \_\_\_

Перевірив:

Пасічник Т.В.

2024

**Завдання:**

Напиcати програми обчислення суми та різниці двох матриць (послідовний та паралельний алгоритми). Порахувати час роботи кожної з програм, обчисліть прискорення та ефективність роботи паралельного алгоритму.

В матрицях розмірності (n,m) робити змінними, щоб легко змінювати величину матриці.

Кількість потоків k - також змінна величина.

Програма повинна показувати час при послідовному способі виконання програми, а також при розпаралеленні на k потоків.

Звернути увагу на випадки, коли розмірність матриці не кратна кількості потоків.

**Програмна реалізація:**

Я використовував мову програмування С# для написання програми. Для вимірів часу роботи використовував бібліотеку System.Diagnostics.

**Робота програми:**

Спочатку програма пропонує вказати кількість рядків та стовпців матриці. Для двох матриць ця кількість є однаковою. Дані генеруються автоматично.

A screenshot of a computer screen

Description automatically generated

Меню дозволяє провести всі дії вручну або ж використати функцію, що автоматично перевіряє оптимальну кількість потоків.

A black background with green text

Description automatically generated

Функція для визначення оптимальної кількості потоків приймає максимальну кількість потоків та відповідно перевіряє кожен варіант від 1-го потоку до максимальної кількості.

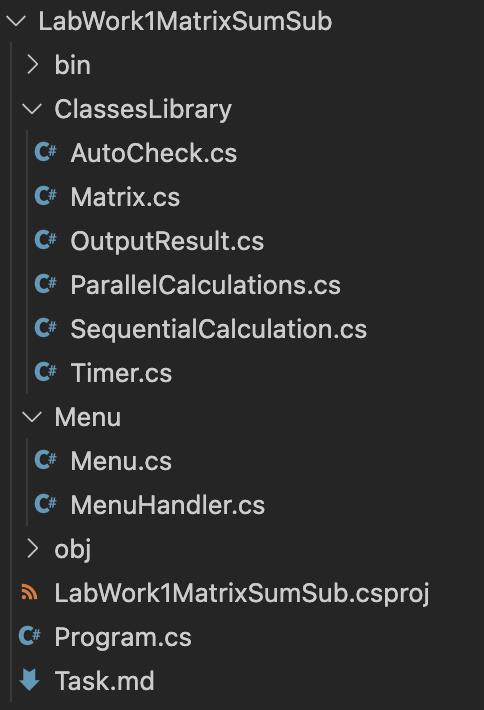
A screen shot of a computer

Description automatically generatedA screen shot of a computer

Description automatically generated

**Структура проєкту:**

Проєкт поділений на декілька частин: бібліотека класів, які використовуються для обрахунків, меню, та основна програма.



**Робота класів:**

Клас `Matrix` представляє матрицю з цілими числами, що має задану кількість рядків і стовпців, і надає методи для роботи з її елементами. Він зберігає дані в масиві `MatrixArray`, розмір якого визначається параметрами `rows` та `columns`. Основні методи класу включають `FillMatrix()`, що заповнює матрицю випадковими числами в діапазоні від 0 до 9, `PrintMatrix()`, що виводить елементи матриці у вигляді таблиці, і `FillZero()`, який встановлює всі елементи матриці в нуль.

Клас SequentialCalculation надає статичні методи для виконання базових операцій з матрицями в послідовному режимі, а саме додавання та віднімання двох матриць.

Клас `ParallelCalculations` забезпечує паралельне виконання операцій додавання та віднімання матриць з використанням багатопоточності. Методи `SumMatrices` і `SubtractMatrices` приймають дві вхідні матриці (`matrix1` та `matrix2`), матрицю результату (`resultMatrix`), а також кількість потоків (`threadsCount`). Для кожного потоку використовується загальний індекс рядків (`currentRow`), який синхронізується за допомогою `lock`, щоб уникнути конфліктів при доступі до рядків. Потоки поперемінно обробляють рядки матриці, додаючи або віднімаючи відповідні елементи вхідних матриць та зберігаючи результати в `resultMatrix`. Такий підхід забезпечує ефективне використання багатопоточності для прискорення обчислень.

Приклад обчислення додавання (віднімання працює аналогічно):

A screen shot of a computer program

Description automatically generated

Клас Timer надає простий інтерфейс для вимірювання часу виконання операцій за допомогою вбудованого об’єкта Stopwatch.

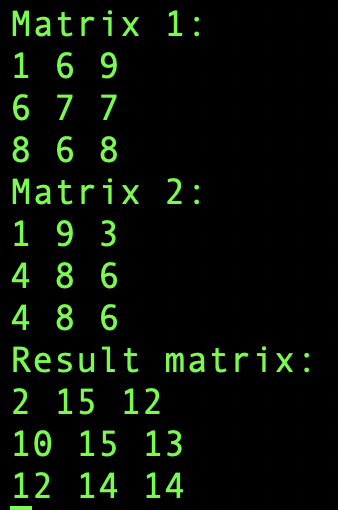
Клас `AutoCheck` виконує автоматичну перевірку для визначення оптимальної кількості потоків, необхідної для додавання двох матриць з найменшим часом виконання. Метод `Check` приймає дві вхідні матриці (`matrix1`, `matrix2`), матрицю результату (`resultMatrix`) та максимальну кількість потоків (`threadsCount`). Спочатку виконується послідовне додавання матриць з використанням одного потоку і фіксується час виконання. Потім метод перевіряє додавання матриць з різною кількістю потоків від 2 до заданого `threadsCount`, порівнюючи час виконання кожного варіанту. У кінці визначається оптимальна кількість потоків, яка забезпечує найменший час виконання, і цей результат виводиться на екран.

Класи Menu та MenuHandler забезпечують зручне користування програмою.

**Дослідження:**

Для початку перевіримо правильність виконання обрахунків на прикладі невеликих матриць.

Для послідовного обчислення:



Для паралельного обчислення:

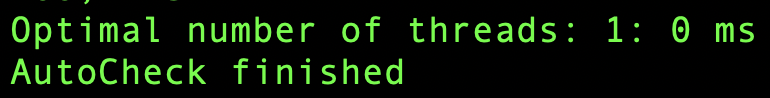
A screenshot of a black screen

Description automatically generated

Результати правильні в обох варіантах.

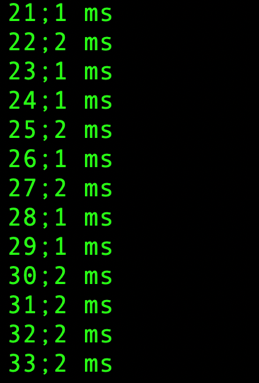
Тепер використовуючи клас AutoCheck, перевіримо як працюють обчислення з різною кількістю потоків для різних розмірів матриць.

Для матриць розміром від 2 на 2 до 100 на 100 оптимальною кількістю потоків був один. Це можна пояснити тим, що такі обчислення є менш ресурсозатратними, ніж розподілення програми на потоки.

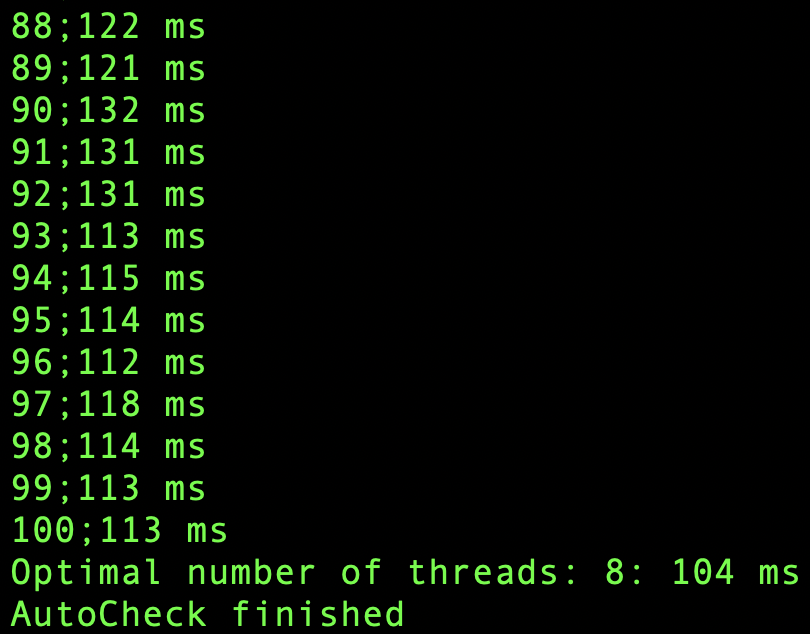
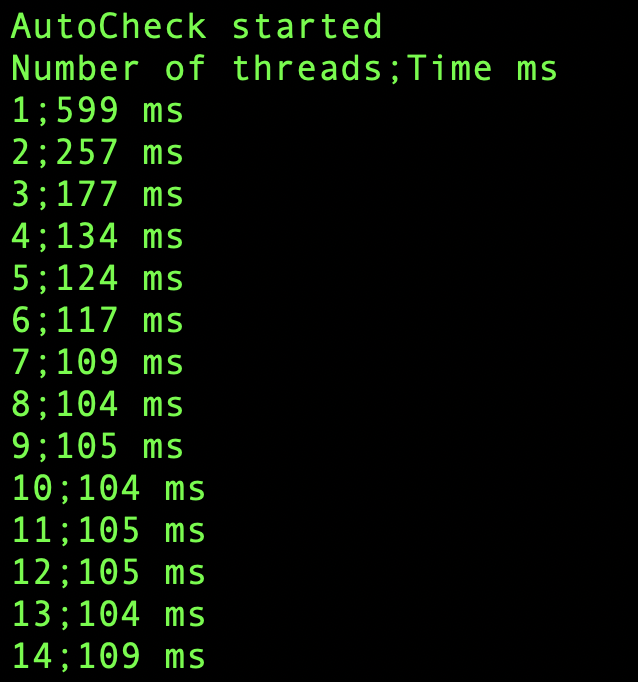


Збільшуючи ж розмір матриць поступово можна побачити, що більша кількість потоків набирає ефективності, оскільки обчислень стає значно більше.

Наприклад, для матриць 500 на 500 оптимальною кількістю потоків є 5. При цьому коли кількість потоків перебільшує 21, то ефективність алгоритму починає спадати.



Тепер розглянемо матриці 7000 на 7000 для того, щоб зрозуміти ефективність розподілення обчислень на потоки.



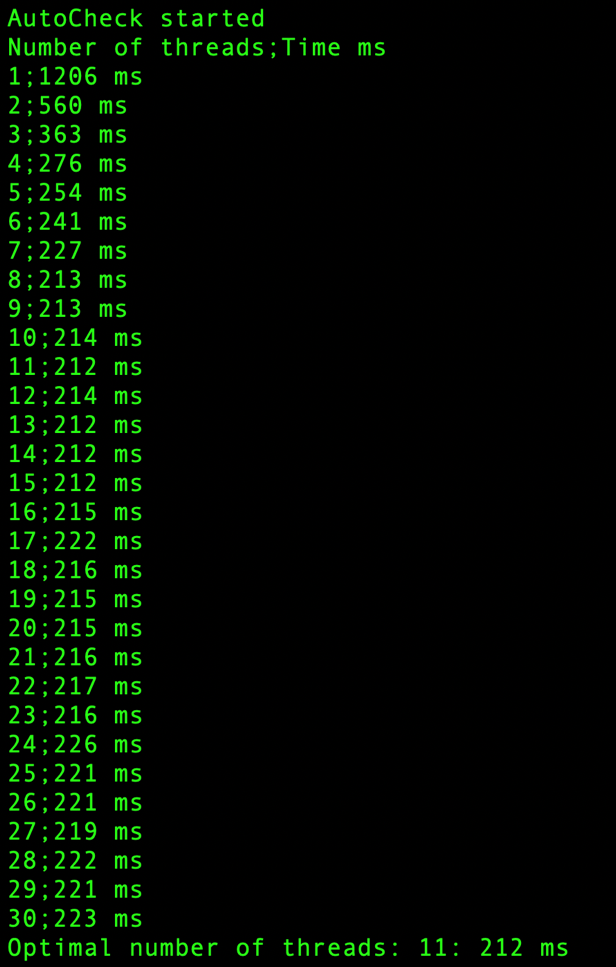
Ефективність є найбільш високою від 8-ми до 14-ти потоків. Мій пристрій має 16-ти потоковий процесор, але програма не може використовувати відразу усі потужності, тому що на фоні є багато інших застосунків, які використовують ресурси, тому можна зрозуміти, чому це справді є близькою до найбільш оптимальної кількості потоків для ефективних обрахунків. Але все ж варто спробувати провести точніші обрахунки.

Для розмірів матриць 10000 на 10000 я провів декілька тестів, щоб знайти середнє значення.

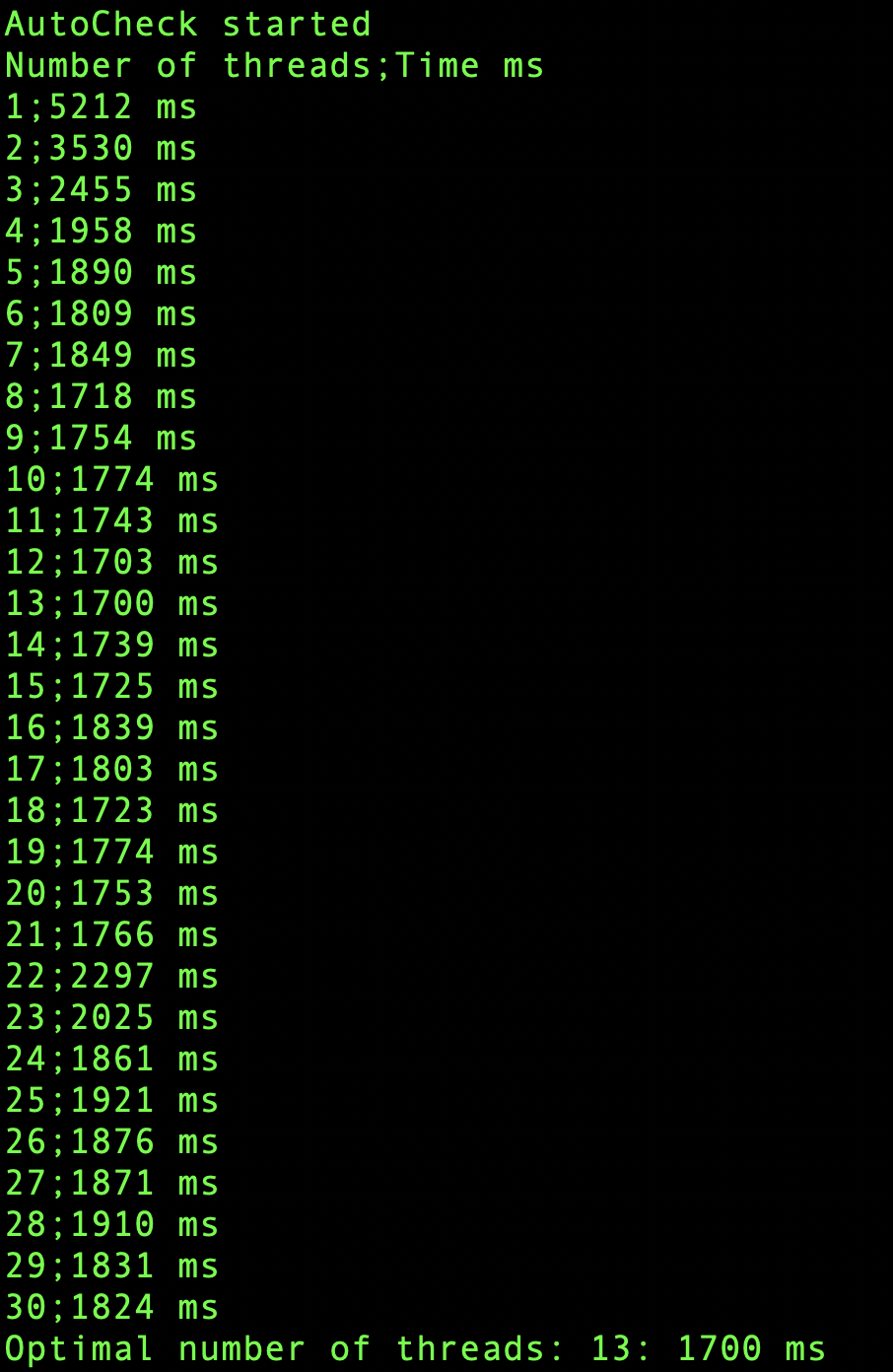
Результат: ( 10 + 9 + 9 + 10 + 9 + 8 + 9 + 8 + 12 + 10) / 10 = 9.4 = 9

Оскільки мій спосіб розбиття на потоки передбачає блокування вже порахованих рядків матриці, то результат не залежить від кратності. Тобто потоки, які будуть «зайвими» просто не виконаються. Це можна перевірити, оглянувши наступні перевірки для матриць 10000 на 10000 та 20000 на 20000 та не знайшовши закономірності між кратністю та швидкістю обрахунків.

10000 на 10000



20000 на 20000



Пізніше додав обрахування прискорення відносно послідовного обчислення та ефективність. Ефективність залежить від прискорення та кількості потоків.

Для 10000 на 10000:

A screenshot of a computer screen

Description automatically generated

Для оптимальної кількості потоків прискорення становить 4,68. Але найбільшою ефективністю характеризується 2-х потокова робота, оскільки співвідношення швидкості та ресурсоефективності у неї найкраще.

Отже, багатопотокові обрахунки здатні зекономити багато часу, обробляючи дані швидко, але варто звертати увагу на ефективність програми та обдумувати, чи варто використовувати усі ресурси, які може запропонувати нам компʼютер.