МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ імені ІВАНА ФРАНКА

**Звіт із**

**Розпаралелення множення матриць**

**Лабораторна робота №2**

з курсу «Паралельні та розподілені обчислення»

Виконав:

Готюк Максим

Група Пмі-33с

Оцінка \_\_\_

Перевірив:

Пасічник Т.В.

2024

**Завдання:**

Написати програми обчислення множення двох матриць (послідовний та паралельний алгоритми). Порахувати час роботи кожної з програм, обчислити прискорення та ефективність роботи паралельного алгоритму.

В матрицях розмірності (n,m) (m,l) зробити 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 матриць. Дані генеруються автоматично.

A screen shot of a computer

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 screenshot of a computer program

Description automatically generatedA screenshot of a computer screen

Description automatically generated

**Структура проєкту:**

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

A screenshot of a computer

Description automatically generated

**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 множення матриць з використанням одного потоку і фіксується час виконання. Потім метод перевіряє множення 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.

Клас ParallelCalculations містить метод MultiplyMatrices, який здійснює паралельне множення двох матриць за допомогою багатопоточності. Метод спочатку перевіряє, чи кількість стовпців першої матриці дорівнює кількості рядків другої матриці, що є обов’язковою умовою для множення матриць. Використовується масив потоків, де кожен потік обробляє різні рядки результатуючої матриці, обчислюючи суму добутків відповідних елементів з рядка першої матриці та стовпця другої. Синхронізація доступу до спільного індексу рядків забезпечується через об’єкт lock, що запобігає конфліктам між потоками. Після завершення роботи всіх потоків об’єднаний результат записується у вихідну матрицю resultMatrix.

A screen shot of a computer program

Description automatically generated

Клас `SequentialCalculation` містить метод `MultiplyMatrices`, який виконує послідовне множення двох матриць без використання багатопоточності. Спочатку перевіряється умова сумісності для множення матриць: кількість стовпців першої матриці повинна дорівнювати кількості рядків другої матриці. Основний алгоритм множення виконується через вкладені цикли: зовнішні два цикли ітерують по рядках першої матриці та стовпцях другої матриці, а внутрішній цикл обчислює суму добутків елементів відповідного рядка та стовпця для обчислення кожного елемента в результатуючій матриці `resultMatrix`. Цей метод виконує базову операцію множення матриць у послідовному режимі, що підходить для обчислень з невеликими обсягами даних, але може бути повільним для великих матриць.

**Дослідження:**

Для початку перевіримо правильність виконання обрахунків на прикладі невеликих матриць.

Для послідовного обчислення:

A screenshot of a computer

Description automatically generated

Для паралельного обчислення:

A screenshot of a cell phone

Description automatically generated

Результати правильні в обох варіантах. Перевірив, використовуючи онлайн калькулятор <https://matrix.reshish.com/multCalculation.php>.

A screenshot of a black and white grid

Description automatically generated

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

Спочатку перевіримо матриці невеликих розмірів. Для матриць розміром 100 на 100 оптимальною кількістю потоків є 7 з прискоренням 14,5.

A screenshot of a computer program

Description automatically generatedA screen shot of a computer program

Description automatically generated

При цьому найвищою ефективністю характеризується процес в 4 потоки: у 2,42 рази ефективніше послідовних обрахунків.

Тепер перевіримо, чи зберігається тенденція для більших матриць.

Для розмірів матриць 1000 на 1000 оптимальною кількістю потоків є 10 з прискоренням у 5,43.

A screenshot of a computer program

Description automatically generatedA screen shot of a computer program

Description automatically generated

При цьому найвищою ефективністю характеризується процес в 2 потоки: в 1,18 разів ефективніше послідовних обрахунків.

Можна спостерігати, що для більших матриць ефективність використання багатопотокових обрахунків є меншою, але вони досі є значно швидшими, ніж послідовні обрахунки.

Варто звернути увагу, що розподілення роботи між потоками є аналогічним, як у додаванні матриць, тому кратність потоків щодо матриці немає значення, бо потоки, яким не вистачило рядків, не запустяться.

Отже, результати показують, що використання багатопотокових обчислень значно підвищує продуктивність операцій з матрицями, зменшуючи час виконання порівняно з послідовними обчисленнями. Прискорення зростає із збільшенням кількості потоків, але після досягнення певної ефективності зростання починає зменшуватися. Це пов'язано з накладними витратами на синхронізацію потоків та управління ресурсами, що свідчить про необхідність оптимального підбору кількості потоків для конкретних задач. Загалом, багатопоточність є ефективним підходом, але її користь знижується при надмірному збільшенні кількості потоків.