

МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ  
ЛЬВІВСЬКИЙ  
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 C# 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.

```
(base) maxpotstakenahs@pcbook: ~/CSharpKzmat1$ dotnet run
Enter the number of columns for the matrice 1 (will apply as row count for 2-nd matrice):
1000
Enter the number of rows for the matrice 1:
500
Enter the number of columns for the matrice 2:
700
```

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

```
0. Exit
1. Multiply matrices sequentially
2. Multiply matrices in parallel
3. Write matrices
4. AutoCheck
```

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

```
Enter the maximum number of threads:
50
AutoCheck started
Number of threads;Time ms;Acceleration; Effectiveness
1;3492 ms
2;1472 ms;2,3722825x;1,1861413x
3;1027 ms;3,4001946x;1,1333982x
4;762 ms;4,5826774x;1,1456693x
5;708 ms;4,9322033x;0,98644066x
6;673 ms;5,1887074x;0,86478454x
7;629 ms;5,551669x;0,7930956x
8;599 ms;5,829716x;0,7287145x
9;596 ms;5,8590603x;0,6510067x
10;598 ms;5,8394647x;0,58394647x
11;603 ms;5,7910447x;0,5264586x
12;596 ms;5,8590603x;0,48825502x
13;600 ms;5,82x;0,44769233x
14;602 ms;5,8006644x;0,41433316x
15;604 ms;5,781457x;0,38543046x
16;608 ms;5,743421x;0,35896382x
17;610 ms;5,7245903x;0,3367406x
18;614 ms;5,6872964x;0,3159609x
19;618 ms;5,6504855x;0,29739398x
20;627 ms;5,569378x;0,2784689x
21;625 ms;5,5872x;0,26605716x
22;626 ms;5,5782747x;0,25355795x
23;635 ms;5,4992127x;0,23909621x
24;627 ms;5,569378x;0,23205741x
25;633 ms;5,5165877x;0,2206635x
26;633 ms;5,5165877x;0,21217646x
27;634 ms;5,5078864x;0,2039958x
28;710 ms;4,9183097x;0,17565392x
29;665 ms;5,2511277x;0,18107337x
30;644 ms;5,4223604x;0,18074535x
31;647 ms;5,3972178x;0,1741038x
32;645 ms;5,4139533x;0,16918604x
33;649 ms;5,3805857x;0,16304804x
34;677 ms;5,15805x;0,15170735x
35;666 ms;5,243243x;0,14980695x
36;656 ms;5,3231707x;0,14786585x
37;655 ms;5,331298x;0,14408913x
38;656 ms;5,3231707x;0,14008343x
39;660 ms;5,2909093x;0,13566434x
40;668 ms;5,227545x;0,13068862x
41;664 ms;5,259036x;0,12826917x
42;740 ms;4,718919x;0,11235521x
43;684 ms;5,105263x;0,11872705x
44;664 ms;5,259036x;0,11952355x
45;668 ms;5,227545x;0,116167665x
46;670 ms;5,2119403x;0,11330305x
47;675 ms;5,173333x;0,11007092x
48;690 ms;5,0608697x;0,10543478x
49;677 ms;5,15805x;0,105266325x
50;673 ms;5,1887074x;0,103774145x
Optimal number of threads: 9: 596 ms
AutoCheck finished
```

## Структура проєкту:

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

```
└─ LabWork2MatrixMult
   │
   │  > bin
   │
   │  └─ ClassesLibrary
   │     │
   │     │  C# AutoCheck.cs
   │     │  C# Matrix.cs
   │     │  C# OutputResult.cs
   │     │  C# ParallelCalculations.cs
   │     │  C# SequentialCalculation.cs
   │     │  C# Timer.cs
   │
   │  └─ Menu
   │     │
   │     │  C# Menu.cs
   │     │  C# MenuHandler.cs
   │
   │  > obj
   │
   │  W ~$it_2_Готюк.docx
   │
   │  LabWork2MatrixMult.csproj
   │
   │  C# Program.cs
   │
   │  Task.md
   │
   │  W Звіт_2_Готюк.docx
```

### 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):

```
public static Matrix SumMatrices(Matrix matrix1, Matrix matrix2, Matrix resultMatrix, int threadsCount)
{
    Thread[] threads = new Thread[threadsCount];
    int currentRow = 0;

    object lockObj = new object();

    for (int i = 0; i < threadsCount; i++)
    {
        threads[i] = new Thread(() =>
        {
            while (true)
            {
                int row;
                lock (lockObj)
                {
                    if (currentRow >= matrix1.Rows)
                        break;
                    row = currentRow++;
                }

                for (int k = 0; k < matrix1.Columns; k++)
                {
                    resultMatrix.MatrixArray[row, k] = matrix1.MatrixArray[row, k] + matrix2.MatrixArray[row, k];
                }
            }
        });
        threads[i].Start();
    }

    foreach (Thread thread in threads)
    {
        thread.Join();
    }
    return resultMatrix;
}
```

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 результуючої 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`.

```
public static Matrix MultiplyMatrices(Matrix matrix1, Matrix matrix2, Matrix resultMatrix, int threadsCount)
{
    if (matrix1.Columns != matrix2.Rows)
    {
        throw new InvalidOperationException("The number of columns in the first matrix must match the number of rows in the second matrix.");
    }
}

You, 38 minutes ago • Wrote multiplication logic.
Thread[] threads = new Thread[threadsCount];
int currentRow = 0;

object lockObj = new object();

for (int i = 0; i < threadsCount; i++)
{
    threads[i] = new Thread(() =>
    {
        while (true)
        {
            int row;
            lock (lockObj)
            {
                if (currentRow >= matrix1.Rows)
                    break;
                row = currentRow++;
            }

            for (int col = 0; col < matrix2.Columns; col++)
            {
                int sum = 0;
                for (int k = 0; k < matrix1.Columns; k++)
                {
                    sum += matrix1.MatrixArray[row, k] * matrix2.MatrixArray[k, col];
                }
                resultMatrix.MatrixArray[row, col] = sum;
            }
        }
    });
    threads[i].Start();
}

foreach (Thread thread in threads)
{
    thread.Join();
}

return resultMatrix;
}
```

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:

```
Matrix 1:
9 8 8
3 2 7
4 6 3
2 4 3
Matrix 2:
4 9 7
8 6 7
7 3 0
Result matrix:
156 153 119
77 60 35
85 81 70
61 51 42
```

Для паралельного обчислення:

```
Matrix 1:
9 8 8
3 2 7
4 6 3
2 4 3
Matrix 2:
4 9 7
8 6 7
7 3 0
Result matrix:
156 153 119
77 60 35
85 81 70
61 51 42
```

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

	C <sub>1</sub>	C <sub>2</sub>	C <sub>3</sub>
1	156	153	119
2	77	60	35
3	85	81	70
4	61	51	42

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.

```

AutoCheck started
Number of threads;Time ms;Acceleration; Effectiveness
1;29 ms
2;9 ms;3,2222223x;1,6111112x
3;5 ms;5,8x;1,9333334x
4;3 ms;9,666667x;2,4166667x
5;3 ms;9,666667x;1,9333334x
6;3 ms;9,666667x;1,6111112x
7;2 ms;14,5x;2,0714285x
8;2 ms;14,5x;1,8125x
9;2 ms;14,5x;1,6111112x
10;2 ms;14,5x;1,45x
11;3 ms;9,666667x;0,87878793x
12;2 ms;14,5x;1,2083334x
13;2 ms;14,5x;1,1153846x
14;2 ms;14,5x;1,0357143x
15;2 ms;14,5x;0,96666664x
16;2 ms;14,5x;0,90625x
17;2 ms;14,5x;0,85294116x
18;2 ms;14,5x;0,8055556x
19;2 ms;14,5x;0,7631579x
20;2 ms;14,5x;0,725x
21;2 ms;14,5x;0,6904762x
22;2 ms;14,5x;0,65909094x
23;2 ms;14,5x;0,6304348x
24;2 ms;14,5x;0,6041667x
25;2 ms;14,5x;0,58x
26;2 ms;14,5x;0,5576923x
27;2 ms;14,5x;0,537037x
28;2 ms;14,5x;0,51785713x
29;2 ms;14,5x;0,5x
30;3 ms;9,666667x;0,32222223x
31;3 ms;9,666667x;0,31182796x
32;2 ms;14,5x;0,453125x
33;3 ms;9,666667x;0,2929293x
34;3 ms;9,666667x;0,28431374x
35;3 ms;9,666667x;0,2761905x
36;3 ms;9,666667x;0,26851854x
37;3 ms;9,666667x;0,26126128x
38;3 ms;9,666667x;0,25438598x
39;3 ms;9,666667x;0,24786326x
40;3 ms;9,666667x;0,24166667x
Optimal number of threads: 7: 2 ms
AutoCheck finished

```

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.

```

AutoCheck started
Number of threads;Time ms;Acceleration; Effectiveness
1;9917 ms
2;4211 ms;2,3550227x;1,1775113x
3;2923 ms;3,3927472x;1,1309158x
4;2224 ms;4,4590826x;1,1147707x
5;2541 ms;3,9027941x;0,7805588x
6;1954 ms;5,07523x;0,8458717x
7;1869 ms;5,306046x;0,7580066x
8;1843 ms;5,380901x;0,6726126x
9;1880 ms;5,275x;0,5861111x
10;1828 ms;5,4250546x;0,54250544x
11;1845 ms;5,3750677x;0,4886425x
12;1849 ms;5,3634396x;0,4469533x
13;1866 ms;5,3145766x;0,4088136x
14;1874 ms;5,291889x;0,3779921x
15;1894 ms;5,2360086x;0,34906724x
16;1951 ms;5,0830345x;0,31768966x
17;2013 ms;4,926478x;0,2897928x
18;1931 ms;5,135681x;0,28531563x
19;1943 ms;5,103963x;0,26862964x
20;1993 ms;4,975916x;0,24879579x
21;2059 ms;4,816416x;0,22935313x
22;1982 ms;5,003532x;0,22743326x
23;1989 ms;4,9859223x;0,21677923x
24;1987 ms;4,990941x;0,20795588x
25;2010 ms;4,9338307x;0,19735323x
26;2457 ms;4,036223x;0,15523934x
27;2244 ms;4,4193406x;0,16367929x
28;2032 ms;4,8804135x;0,17430048x
29;2033 ms;4,8780127x;0,16820733x
30;2034 ms;4,8756146x;0,16252048x
31;2059 ms;4,816416x;0,15536825x
32;2069 ms;4,7931366x;0,14978552x
33;2090 ms;4,744976x;0,14378715x
34;2149 ms;4,6147046x;0,1357266x
35;2084 ms;4,7586374x;0,13596107x
36;2094 ms;4,7359123x;0,13155311x
37;2103 ms;4,7156444x;0,12744984x
38;2105 ms;4,711164x;0,123978x
39;2111 ms;4,6977735x;0,12045573x
40;2256 ms;4,3958335x;0,10989584x
Optimal number of threads: 10: 1828 ms
AutoCheck finished

```

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 запускаються.

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.