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:

Написати 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 C# 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.

```
Enter the number of rows for the matrices:
7000
Enter the number of columns for the matrices:
7000
```

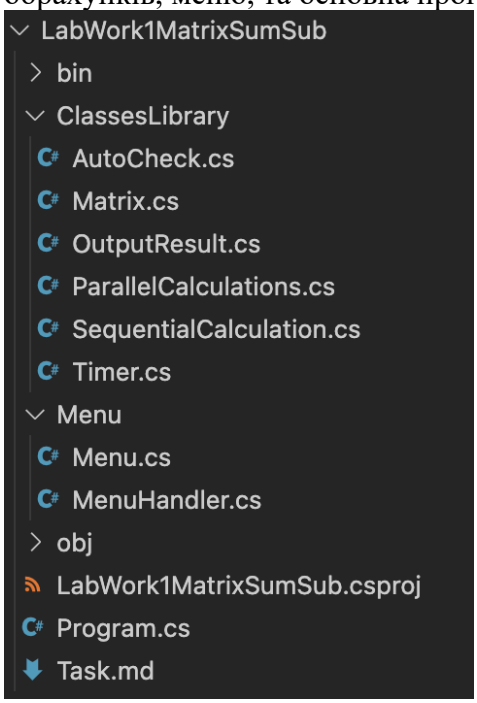
Меню дозволяє провести всі дії вручну або ж використати функцію, що автоматично перевіряє оптимальну кількість потоків.

```
0. Exit
1. Sum matrices sequentially
2. Subtract matrices sequentially
3. Sum matrices in parallel
4. Subtract matrices in parallel
5. Write matrices
6. AutoCheck
```

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

```
6
Enter the maximum number of threads:
100
AutoCheck started
Number of threads;Time ms
1;649 ms
2;257 ms
3;177 ms
4;138 ms
5;124 ms
6;116 ms
7;109 ms
8;104 ms
9;104 ms
10;104 ms
11;104 ms
12;105 ms
13;105 ms
14;105 ms
15;106 ms
16;106 ms
17;106 ms
18;107 ms
86;115 ms
87;117 ms
88;119 ms
89;116 ms
90;127 ms
91;139 ms
92;120 ms
93;123 ms
94;135 ms
95;134 ms
96;127 ms
97;127 ms
98;123 ms
99;117 ms
100;118 ms
Optimal number of threads: 8: 104 ms
AutoCheck finished
```

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):

```
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++;
                }      You, 44 minutes ago • LabWork program done.

                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 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:

```
Matrix 1:
1 6 9
6 7 7
8 6 8
Matrix 2:
1 9 3
4 8 6
4 8 6
Result matrix:
2 15 12
10 15 13
12 14 14
```

Для паралельного обчислення:

```
Matrix 1:
1 6 9
6 7 7
8 6 8
Matrix 2:
1 9 3
4 8 6
4 8 6
Result matrix:
2 15 12
10 15 13
12 14 14
```

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.

```
Optimal number of threads: 1: 0 ms
AutoCheck finished
```

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.

```
AutoCheck started
Number of threads;Time ms
1;9 ms
2;4 ms
3;2 ms
4;2 ms
5;1 ms
6;1 ms
7;1 ms
8;1 ms
9;1 ms
21;1 ms
22;2 ms
23;1 ms
24;1 ms
25;2 ms
26;1 ms
27;2 ms
28;1 ms
29;1 ms
30;2 ms
31;2 ms
32;2 ms
33;2 ms
```

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

```
AutoCheck started
Number of threads;Time ms
1;599 ms
2;257 ms
3;177 ms
4;134 ms
5;124 ms
6;117 ms
7;109 ms
8;104 ms
9;105 ms
10;104 ms
11;105 ms
12;105 ms
13;104 ms
14;109 ms
88;122 ms
89;121 ms
90;132 ms
91;131 ms
92;131 ms
93;113 ms
94;115 ms
95;114 ms
96;112 ms
97;118 ms
98;114 ms
99;113 ms
100;113 ms
Optimal number of threads: 8: 104 ms
AutoCheck finished
```

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

```
AutoCheck started
Number of threads;Time ms
1;1206 ms
2;560 ms
3;363 ms
4;276 ms
5;254 ms
6;241 ms
7;227 ms
8;213 ms
9;213 ms
10;214 ms
11;212 ms
12;214 ms
13;212 ms
14;212 ms
15;212 ms
16;215 ms
17;222 ms
18;216 ms
19;215 ms
20;215 ms
21;216 ms
22;217 ms
23;216 ms
24;226 ms
25;221 ms
26;221 ms
27;219 ms
28;222 ms
29;221 ms
30;223 ms
Optimal number of threads: 11: 212 ms
```

20000 на 20000

```
AutoCheck started
Number of threads;Time ms
1;5212 ms
2;3530 ms
3;2455 ms
4;1958 ms
5;1890 ms
6;1809 ms
7;1849 ms
8;1718 ms
9;1754 ms
10;1774 ms
11;1743 ms
12;1703 ms
13;1700 ms
14;1739 ms
15;1725 ms
16;1839 ms
17;1803 ms
18;1723 ms
19;1774 ms
20;1753 ms
21;1766 ms
22;2297 ms
23;2025 ms
24;1861 ms
25;1921 ms
26;1876 ms
27;1871 ms
28;1910 ms
29;1831 ms
30;1824 ms
Optimal number of threads: 13: 1700 ms
```

Пізніше додав обрахування прискорення відносно послідовного обчислення та ефективність. Ефективність залежить від прискорення та кількості потоків.
Для 10000 на 10000:

```
Number of threads;Time ms;Acceleration; Effectiveness
1;1289 ms
2;525 ms;2,455238x;1,227619x
3;363 ms;3,550964x;1,1836547x
4;272 ms;4,7389708x;1,1847427x
5;251 ms;5,135458x;1,0270916x
6;240 ms;5,3708334x;0,8951389x
7;221 ms;5,832579x;0,8332256x
8;214 ms;6,0233645x;0,75292057x
9;214 ms;6,0233645x;0,6692627x
10;216 ms;5,9675927x;0,59675926x
11;214 ms;6,0233645x;0,5475786x
12;218 ms;5,912844x;0,49273703x
13;212 ms;6,0801888x;0,46770683x
14;213 ms;6,0516434x;0,43226025x
15;213 ms;6,0516434x;0,4034429x
16;214 ms;6,0233645x;0,37646028x
17;213 ms;6,0516434x;0,35597903x
18;219 ms;5,8858447x;0,32699138x
19;215 ms;5,995349x;0,3155447x
20;216 ms;5,9675927x;0,29837963x
21;217 ms;5,940092x;0,28286153x
22;220 ms;5,859091x;0,2663223x
23;218 ms;5,912844x;0,25708017x
24;218 ms;5,912844x;0,24636851x
25;217 ms;5,940092x;0,23760368x
26;217 ms;5,940092x;0,22846508x
27;218 ms;5,912844x;0,21899423x
28;217 ms;5,940092x;0,21214615x
29;218 ms;5,912844x;0,20389117x
30;219 ms;5,8858447x;0,19619483x
31;219 ms;5,8858447x;0,18986596x
32;222 ms;5,8063064x;0,18144707x
33;221 ms;5,832579x;0,17674482x
34;220 ms;5,859091x;0,1723262x
35;227 ms;5,678414x;0,1622404x
36;226 ms;5,70354x;0,15843166x
37;235 ms;5,4851065x;0,14824612x
38;234 ms;5,508547x;0,14496176x
39;244 ms;5,282787x;0,13545607x
40;228 ms;5,6535087x;0,14133772x
41;225 ms;5,728889x;0,139729x
42;225 ms;5,728889x;0,13640212x
43;228 ms;5,6535087x;0,13147694x
44;226 ms;5,70354x;0,1296259x
45;233 ms;5,532189x;0,12293753x
46;230 ms;5,6043477x;0,121833645x
47;232 ms;5,5560346x;0,118213505x
48;234 ms;5,508547x;0,11476139x
49;230 ms;5,6043477x;0,114374444x
50;232 ms;5,5560346x;0,11112069x
Optimal number of threads: 13: 212 ms
```

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 комп’ютер.