**Міністерство освіти і науки України  
Національний технічний університет України  
«Київський політехнічний інститут імені Ігоря Сікорського»  
Факультет інформатики та обчислювальної техніки  
Кафедра обчислювальної техніки**

**Лабораторна робота №3**

«Програмування для комп’ютерних систем зі спільною пам’яттю. Семафори, мютекси, події, критичні секції, атомік-змінні, бар’єри»

з дисципліни  
«Програмне забезпечення високопродуктивних комп’ютерних систем»

Виконала:

студентка групи ІМ-11   
Бащак Ярина Володимирівна

Перевірив:

доц. Корочкін О. В.

Київ 2024

**Завдання.**

Побудувати паралельний алгоритм для обчислення формули. Розробити паралельну програму за допомогою мови C#, яка б виконувала це паралельне обчислення. Формула, схема введення та виведення даних потоками обирається згідно варіанту.

Програма має містити класи Lab3, Data та потоки T1, T2, T3, T4.

**Варіант 20:**

Завдання: X = (B\*Z)\*(d\*Z + R\*(MO\*MR))

|  |  |  |  |
| --- | --- | --- | --- |
| Введення – виведення даних | | | |
| 1 | 2 | 3 | 4 |
| MO | Z, R | B, MR | X, d |

**Виконання роботи.**

**Етап 1. Побудова паралельного математичного алгоритму.**

Розроблено наступний алгоритм:

1. ai = Bн\*Zн, де Bн, Zн - це H елементів відповідних векторів;
2. a = a + ai; СР: a;
3. Xн = a\*(d\*Zн + R\*(MO\*MRн)) СР: a, d;  
   де Xн і Zн - H елементів відповідних векторів, а MRн - Н стовпців матриці MR.

Серед вказаних спільних ресурсів захист потрібен:

* скаляру a при перезаписі і копіюванні;
* скаляру d при копіюванні.

**Етап 2. Розробка алгоритмів потоків.**

Далі наведено покровий алгоритм для кожної з чотирьох задач.

Задача T1 Точки синхронізації

1. Ведення MO
2. Сигнал задачам T2, T3, T4 про введення MO -- S2-1, S3-1, S4-1
3. Чекати на введення даних в задачах T2, T3, T4 -- W2-1, W3-1, W4-1
4. Обчислення 1: a1 = Bн\*Zн
5. Обчислення 2: a = a + a1 -- КД1
6. Сигнал задачам T2, T3, T4 про завершення обчислення 2 -- S2-2, S3-2, S4-2
7. Чекати на завершення обчислення 2 в задачах T2, T3, T4 -- W2-2, W3-2, W4-2
8. Копіювання d1 = d -- КД2
9. Копіювання a1 = a -- КД3
10. Обчислення 3: Xн = a1\*(d1\*Zн + R\*(MO\*MRн))
11. Сигнал задачі T4 про завершення обчислення 3 -- S4-3

Задача T2 Точки синхронізації

1. Введення Z, R
2. Сигнал задачам T1, T3, T4 про введення Z, R -- S1-1, S3-1, S4-1
3. Чекати на введення даних в задачах T1, T3, T4 -- W1-1, W3-1, W4-1
4. Обчислення 1: a2 = Bн\*Zн
5. Обчислення 2: a = a + a2 -- КД1
6. Сигнал задачам T1, T3, T4 про завершення обчислення 2 -- S1-2, S3-2, S4-2
7. Чекати на завершення обчислення 2 в задачах T1, T3, T4 -- W1-2, W3-2, W4-2
8. Копіювання d2 = d -- КД2
9. Копіювання a2 = a -- КД3
10. Обчислення 3: Xн = a2\*(d2\*Zн + R\*(MO\*MRн))
11. Сигнал задачі T4 про завершення обчислення 3 -- S4-3

Задача T3 Точки синхронізації

1. Введення B, MR
2. Сигнал задачам T1, T2, T4 про введення Z, R -- S1-1, S2-1, S4-1
3. Чекати на введення даних в задачах T1, T2, T4 -- W1-1, W2-1, W4-1
4. Обчислення 1: a3 = Bн\*Zн
5. Обчислення 2: a = a + a3 -- КД1
6. Сигнал задачам T1, T2, T4 про завершення обчислення 2 -- S1-2, S2-2, S4-2
7. Чекати на завершення обчислення 2 в задачах T1, T2, T4 -- W1-2, W2-2, W4-2
8. Копіювання d3 = d -- КД2
9. Копіювання a3 = a -- КД3
10. Обчислення 3: Xн = a3\*(d3\*Zн + R\*(MO\*MRн))
11. Сигнал задачі T4 про завершення обчислення 3 -- S4-3

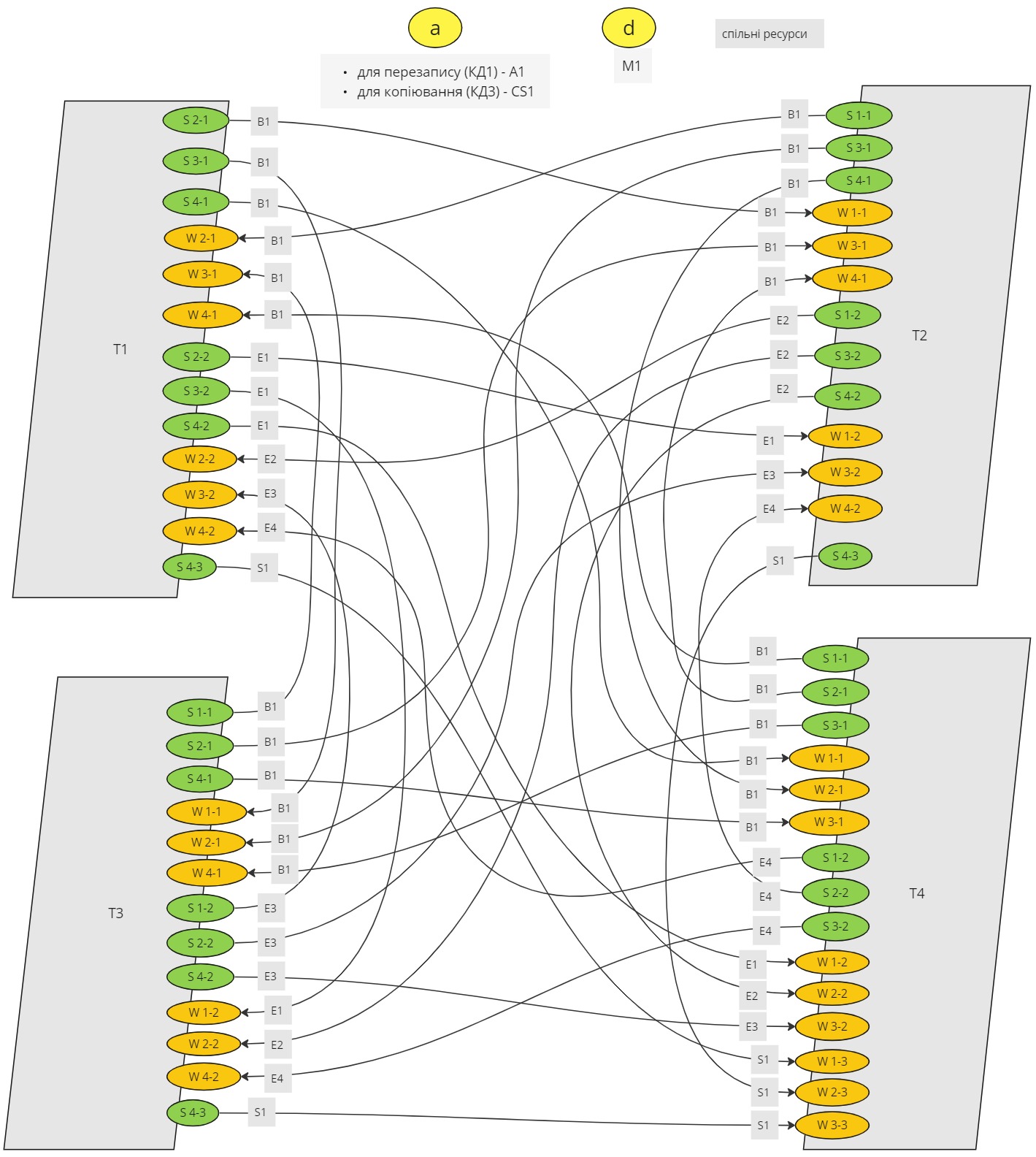
Задача T4 Точки синхронізації

1. Введення d
2. Сигнал задачам T1, T2, T3 про введення d -- S1-1, S2-1, S3-1
3. Чекати на введення даних в задачах T1, T2, T3 -- W1-1, W2-1, W3-1
4. Обчислення 1: a4 = Bн\*Zн
5. Обчислення 2: a = a + a4 -- КД1
6. Сигнал задачам T1, T2, T3 про завершення обчислення 2 -- S1-2, S2-2, S3-2
7. Чекати на завершення обчислення 2 в задачах T1, T2, T3 -- W1-2, W2-2, W3-2
8. Копіювання d4 = d -- КД2
9. Копіювання a4 = a -- КД3
10. Обчислення 3: Xн = a4\*(d4\*Zн + R\*(MO\*MRн))
11. Чекати на завершення обчислення 3 в задачах T1, T2, T3 -- W1-3, W2-3, W3-3
12. Виведення результату X

**Етап 3. Розробка схеми взаємодії задач**

При розробці структурної схеми взаємодії задач, що наведена на мал. 1, було враховано які засоби організації синхронізації реалізовані в мові C#, а саме семафори, мютекси, події, критичні секції, атомік-змінні і бар’єри. Далі наведено опис використання кожного з них:

* бар’єр В1 для синхронізації введення даних з потоків Т1, Т2, Т3, Т4;
* атомік-змінна А1 для реалізації КД1 (перезапис спільного ресурсу);
* події Е1, Е2, Е3, Е4 для сигналів про завершення обчислення 2;
* мютекс М1 для реалізації КД2 (копіювання спільного ресурсу);
* критична секція CS1 для реалізації КД3 (копіювання спільного ресурсу);
* семафор S1 для сигналів до Т4 про завершення задачами обчислення 3.



Мал. 1 Схема взаємодії задач

**Етап 4. Розроблення програми.**

На основі розроблених алгоритмів на етапі 2 і структурної схеми взаємодії задач на етапі 3 було реалізовано програмі на мові C#.

Програма складається з основного класу Lab3, класу Data, що зберігає всі спільні змінні та містить статичні методи для роботи над векторами і матрицями, а також функції T1, T2, T3, T4 для відповідних потоків. Лістинг програми представлено далі.

// Програмне забезпечення високопродуктивних комп'ютерних систем

// Лабораторна робота №3: програмування для КС із СП. Семафори, мютекси, події, критичні секції, атомік-змінні, бар’єри

// варіант 20

// X = (B\*Z)\*(d\*Z + R\*(MO\*MR))

// Бащак Ярина Володимирівна

// група ІМ-11

// 28.03.2024

using System.Diagnostics;

using lab3;

class Lab3

{

    private static readonly Data data = new Data();

    private Stopwatch stopwatch = new Stopwatch();

    static void Main(string[] args)

    {

        Lab3 lab3Instance = new Lab3();

        lab3Instance.stopwatch.Start();

        var t1 = new Thread(new ParameterizedThreadStart(lab3Instance.T1));

        var t2 = new Thread(new ParameterizedThreadStart(lab3Instance.T2));

        var t3 = new Thread(new ParameterizedThreadStart(lab3Instance.T3));

        var t4 = new Thread(new ParameterizedThreadStart(lab3Instance.T4));

        t1.Start(1);

        t2.Start(2);

        t3.Start(3);

        t4.Start(4);

        t1.Join();

        t2.Join();

        t3.Join();

        t4.Join();

    }

    void T1(object param)

    {

        int threadId = (int)param;

        int a1, d1;

        Console.WriteLine($"T{threadId} start");

        // введення МО

        data.FillDataT1();

        // синхронізація введення за допомогою бар'єру B1

        data.B1.SignalAndWait();

        // обчислення 1: a1 = Bн\*Zн

        a1 = Data.MultiplyVectors(Data.GetSubvector(data.B, threadId), Data.GetSubvector(data.Z, threadId));

        // обчислення 2: а = а + а1 - КД1 - атомік-змінна (атомарна операція)

        Interlocked.Add(ref data.a, a1);

        // сигнал про завершення обчислення 2 - подія Е1

        data.E1.Set();

        // чекати, щоб всі потоки завершили обчислення 2 - події Е2, Е3, Е4

        data.E2.WaitOne();

        data.E3.WaitOne();

        data.E4.WaitOne();

        // захист скаляру d за допомогою мютексу М1 - КД2

        data.M1.WaitOne();

        d1 = data.d;

        data.M1.ReleaseMutex();

        // захист скаляру а за допомогою критичної секції - КД3

        lock (data.\_lockObject)

        {

            a1 = data.a;

        }

        // обчислення 3: Xн = a1\*(d1\*Zн + R\*(MO\*MRн))

        data.CalculateStep3(a1, d1, threadId);

        // сигнал задачі Т4 про завершення обчислення 3 - семафор S1

        data.S1.Release();

        Console.WriteLine($"T{threadId} finish");

    }

    void T2(object param)

    {

        int threadId = (int)param;

        int a2, d2;

        Console.WriteLine($"T{threadId} start");

        // введення Z, R

        data.FillDataT2();

        // синхронізація введення за допомогою бар'єру B1

        data.B1.SignalAndWait();

        // обчислення 1: a2 = Bн\*Zн

        a2 = Data.MultiplyVectors(Data.GetSubvector(data.B, threadId), Data.GetSubvector(data.Z, threadId));

        // обчислення 2: а = а + а2 - КД1 - атомік-змінна (атомарна операція)

        Interlocked.Add(ref data.a, a2);

        // сигнал про завершення обчислення 2 - подія Е2

        data.E2.Set();

        // чекати, щоб всі потоки завершили обчислення 2 - події Е1, Е3, Е4

        data.E1.WaitOne();

        data.E3.WaitOne();

        data.E4.WaitOne();

        // захист скаляру d за допомогою мютексу М1 - КД2

        data.M1.WaitOne();

        d2 = data.d;

        data.M1.ReleaseMutex();

        // захист скаляру а за допомогою критичної секції - КД3

        lock (data.\_lockObject)

        {

            a2 = data.a;

        }

        // обчислення 3: Xн = a2\*(d2\*Zн + R\*(MO\*MRн))

        data.CalculateStep3(a2, d2, threadId);

        // сигнал задачі Т4 про завершення обчислення 3 - семафор S1

        data.S1.Release();

        Console.WriteLine($"T{threadId} finish");

    }

    void T3(object param)

    {

        int threadId = (int)param;

        int a3, d3;

        Console.WriteLine($"T{threadId} start");

        // введення B, MR

        data.FillDataT3();

        // синхронізація введення за допомогою бар'єру B1

        data.B1.SignalAndWait();

        // обчислення 1: a3 = Bн\*Zн

        a3 = Data.MultiplyVectors(Data.GetSubvector(data.B, threadId), Data.GetSubvector(data.Z, threadId));

        // обчислення 2: а = а + а3 - КД1 - атомік-змінна (атомарна операція)

        Interlocked.Add(ref data.a, a3);

        // сигнал про завершення обчислення 2 - подія Е3

        data.E3.Set();

        // чекати, щоб всі потоки завершили обчислення 2 - події Е1, Е2, Е4

        data.E1.WaitOne();

        data.E2.WaitOne();

        data.E4.WaitOne();

        // захист скаляру d за допомогою мютексу М1 - КД2

        data.M1.WaitOne();

        d3 = data.d;

        data.M1.ReleaseMutex();

        // захист скаляру а за допомогою критичної секції - КД3

        lock (data.\_lockObject)

        {

            a3 = data.a;

        }

        // обчислення 3: Xн = a3\*(d3\*Zн + R\*(MO\*MRн))

        data.CalculateStep3(a3, d3, threadId);

        // сигнал задачі Т4 про завершення обчислення 3 - семафор S1

        data.S1.Release();

        Console.WriteLine($"T{threadId} finish");

    }

    void T4(object param)

    {

        int threadId = (int)param;

        int a4, d4;

        Console.WriteLine($"T{threadId} start");

        // введення d

        data.FillDataT4();

        // синхронізація введення за допомогою бар'єру B1

        data.B1.SignalAndWait();

        // обчислення 1: a4 = Bн\*Zн

        a4 = Data.MultiplyVectors(Data.GetSubvector(data.B, threadId), Data.GetSubvector(data.Z, threadId));

        // обчислення 2: а = а + а4 - КД1 - атомік-змінна (атомарна операція)

        Interlocked.Add(ref data.a, a4);

        // сигнал про завершення обчислення 2 - подія Е4

        data.E4.Set();

        // чекати, щоб всі потоки завершили обчислення 2 - події E1, Е2, Е3

        data.E1.WaitOne();

        data.E2.WaitOne();

        data.E3.WaitOne();

        // захист скаляру d за допомогою мютексу М1 - КД2

        data.M1.WaitOne();

        d4 = data.d;

        data.M1.ReleaseMutex();

        // захист скаляру а за допомогою критичної секції - КД3

        lock (data.\_lockObject)

        {

            a4 = data.a;

        }

        // обчислення 3: Xн = a4\*(d4\*Zн + R\*(MO\*MRн))

        data.CalculateStep3(a4, d4, threadId);

        // чекати, щоб всі потоки завершили обчислення 3 - семафор S1

        for (int i = 0; i < 3; i++)

        {

            data.S1.Wait();

        }

        stopwatch.Stop();

        // виведення результату Х

        Data.PrintVector(data.X);

        Console.WriteLine($"T{threadId} finish");

        Console.WriteLine($"Time taken: {stopwatch.ElapsedMilliseconds} ms");

        data.B1.Dispose();

    }

}

namespace lab3;

using System.Threading;

public class Data

{

    public static int N { get; set; } = 2000;

    public static int P { get; set; } = 4;

    public static int H = N / P;

    public int[,] MO { get; set; } = new int[N, N];

    public int[,] MR { get; set; } = new int[N, N];

    public int[] Z { get; set; } = new int[N];

    public int[] R { get; set; } = new int[N];

    public int[] B { get; set; } = new int[N];

    public long[] X { get; set; } = new long[N];

    public int d { get; set; }

    public int a = 0;

    public Barrier B1 = new Barrier(4);

    public EventWaitHandle E1 = new EventWaitHandle(false, EventResetMode.ManualReset);

    public EventWaitHandle E2 = new EventWaitHandle(false, EventResetMode.ManualReset);

    public EventWaitHandle E3 = new EventWaitHandle(false, EventResetMode.ManualReset);

    public EventWaitHandle E4 = new EventWaitHandle(false, EventResetMode.ManualReset);

    public Mutex M1 = new Mutex();

    public SemaphoreSlim S1 = new SemaphoreSlim(0, 3);

    public object \_lockObject = new object();

    public void FillDataT1()

    {

        for (int i = 0; i < N; i++)

        {

            for (int j = 0; j < N; j++)

            {

                MO[i, j] = 1;

            }

        }

    }

    public void FillDataT2()

    {

        for (int i = 0; i < N; i++)

        {

            Z[i] = 1;

            R[i] = 1;

        }

    }

    public void FillDataT3()

    {

        for (int i = 0; i < N; i++)

        {

            B[i] = 1;

            for (int j = 0; j < N; j++)

            {

                MR[i, j] = 1;

            }

        }

    }

    public void FillDataT4()

    {

        d = 1;

    }

    public static int[,] MultiplyMatrices(int[,] matrix1, int[,] matrix2)

    {

        int m = matrix1.GetLength(0);

        int n = matrix2.GetLength(0);

        int p = matrix2.GetLength(1);

        int[,] result = new int[m, p];

        for (int i = 0; i < m; i++)

        {

            for (int j = 0; j < p; j++)

            {

                result[i, j] = 0;

                for (int k = 0; k < n; k++)

                {

                    result[i, j] += matrix1[i, k] \* matrix2[k, j];

                }

            }

        }

        return result;

    }

    public static int MultiplyVectors(int[] vector1, int[] vector2)

    {

        int result = 0;

        for (int i = 0; i < vector1.Length; i++)

        {

            result += vector1[i] \* vector2[i];

        }

        return result;

    }

    public static int[] MultiplyVectorByMatrix(int[] vector, int[,] matrix)

    {

        int matrixRows = matrix.GetLength(0);

        int matrixColumns = matrix.GetLength(1);

        int[] result = new int[matrixColumns];

        for (int j = 0; j < matrixColumns; j++)

        {

            result[j] = 0;

            for (int i = 0; i < matrixRows; i++)

            {

                result[j] += vector[i] \* matrix[i, j];

            }

        }

        return result;

    }

    public static int[] MultiplyVectorByScalar(int[] vector, int scalar)

    {

        int[] result = new int[vector.Length];

        for (int i = 0; i < vector.Length; i++)

        {

            result[i] = vector[i] \* scalar;

        }

        return result;

    }

    public static long[] MultiplyVectorByScalar(long[] vector, int scalar)

    {

        long[] result = new long[vector.Length];

        for (int i = 0; i < vector.Length; i++)

        {

            result[i] = vector[i] \* scalar;

        }

        return result;

    }

    public static int[] AddVectors(int[] vector1, int[] vector2)

    {

        int[] result = new int[vector1.Length];

        for (int i = 0; i < vector1.Length; i++)

        {

            result[i] = vector1[i] + vector2[i];

        }

        return result;

    }

    public static int[,] GetSubmatrixFromColumns(int[,] matrix, int threadId)

    {

        int start = Start(threadId);

        int end = End(threadId);

        int rows = matrix.GetLength(0);

        int columns = end - start;

        int[,] submatrix = new int[rows, columns];

        for (int i = 0; i < rows; i++)

        {

            for (int j = start; j < end; j++)

            {

                submatrix[i, j - start] = matrix[i, j];

            }

        }

        return submatrix;

    }

    public static int[] GetSubvector(int[] vector, int threadId)

    {

        int start = Start(threadId);

        int end = End(threadId);

        return vector.Skip(start).Take(end - start).ToArray();

    }

    public static void InsertSubvectorIntoVector(long[] targetVector, long[] subvector, int threadId)

    {

        int start = Start(threadId);

        int size = subvector.GetLength(0);

        for (int i = 0; i < size; i++)

        {

            targetVector[start + i] = subvector[i];

        }

    }

    public static void PrintVector(long[] vector)

    {

        for (int i = 0; i < vector.Length; i++)

        {

            Console.Write(vector[i] + " ");

        }

        Console.WriteLine();

    }

    public void CalculateStep3(int ai, int di, int threadId)

    {

        int[] Z\_H = GetSubvector(Z, threadId);

        int[,] MR\_H = GetSubmatrixFromColumns(MR, threadId);

        int[] di\_Z\_H = MultiplyVectorByScalar(Z\_H, di);

        int[,] MO\_MR\_H = MultiplyMatrices(MO, MR\_H);

        int[] R\_MO\_MR\_H = MultiplyVectorByMatrix(R, MO\_MR\_H);

        int[] di\_Z\_H\_plus\_R\_MO\_MR\_H = AddVectors(di\_Z\_H, R\_MO\_MR\_H);

        long[] X\_H = MultiplyVectorByScalar(di\_Z\_H\_plus\_R\_MO\_MR\_H.Select(x => (long)x).ToArray(), ai);

        InsertSubvectorIntoVector(X, X\_H, threadId);

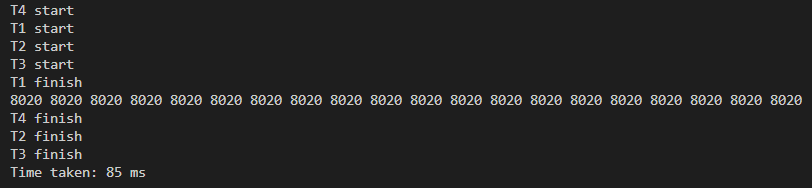
    }

    private static int Start(int threadId) => (threadId - 1) \* H;

    private static int End(int threadId) => threadId \* H;

}

На мал. 2 надано скриншот результату роботи програми для N = 20.



Мал. 2 Вивід програми

**Етап 5. Знаходження коефіцієнту прискорення.**

Тестування програми було проведено на комп’ютері з процесором Intel(R) Core(TM) i5-8265U, ядер 4, логічних процесорів 8.

У рамках експерименту з обчисленням ефективності багатопоточності програми при N=2000, було встановлено, що:

* час виконання програми, коли вона розподіляється системою автоматично по доступним ядрам: 108,8 секунд;
* час при обмеженні виконання лише 1-м ядром: 156,7 секунда.

Отже, Кп = 156,7/108,8 = 1,44.

**Висновки:**

1. розроблено паралельний математичний алгоритм, який дозволяє виконати обчислення заданої формули і визначити, що спільними ресурсами є скаляри a (перезапис і копіювання) і d (тільки копіювання);
2. розроблений алгоритм кожної з задач, визначені точки синхронізації, до яких входять синхронізація по введенню даних, виконанню обчислення 3 і виведенню результату, а також три завдання взаємного виключення пов'язаних з перезаписом або копіюванням спільних ресурсів;
3. побудовано структурну схему їх взаємодії, обрано такі засоби синхронізації:
   * бар’єр, щоб синхронізувати введення даних;
   * атомік-змінну, щоб забезпечити атомарність операції перезапису;
   * події ручного скидання, щоб надсилати і приймати сигнали про завершення обчислення 2;
   * мютекс, щоб контролювати копіювання спільного ресурсу d;
   * критичну секцію, щоб безпечно копіювати спільний ресурс a;
   * семафор, щоб синхронізувати виведення результату;
4. реалізовано багатопоточну програму на мові C#, де для кожної задачі створюється окремий потік за допомогою конструктора класу Thread і відповідної потокової функції;
5. проведено тестування, яке показало ефективність багатопоточної програми: при N = 2000 Кп = 1,44.