**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 №2
# «Семафори, мютекси, події, критичні секції»

з дисципліни
«Програмне забезпечення високопродуктивних комп'ютерних систем»

Виконала:
Студентка групи ІМ-13
Дубчак Аліна Євгеніївна
номер у списку групи: 5

Перевірив:
Корочкін О. В.

Київ 2024

# ПОСТАНОВКА ЗАДАЧІ

- розробити паралельний алгоритм рішення математичної задачі;
- виявити спільні ресурси;
- описати алгоритм кожного потоку (T1 – Tp ) з визначенням критичних ділянок (КД) і точок синхронізації (Wij , Sij);
- розробити структурну схему взаємодії задач (Див. Додаток А), де застосувати ВСІ вказані засоби взаємодії процесів
- розробити програму (обов'язкові "шапка", коментарі )
- виконати налагодження програми;
- отримати правильні результати обчислень.
- за допомогою Диспетчеру задач Windows проконтролювати завантаження ядер процесору.
- введення даних повинно бути через забивання всіх елементів 1

**Варіант завдання:**

X = (B*Z)*(d*Z + R*(MO*MR))
1 – MO
2 – Z, R
3 – B, MR
4 – X, d

# ОПИС ПРОГРАМИ

**І етап. Побудова паралельного математичного алгоритму**

**Варіант 20**
X = (B*Z)*(d*Z + R*(MO*MR))

1) ai = Bн * Zн i= 1...4
2) a = a + ai CP:a, копія
3) MEн = MRн* MO CP: MO (ігноруємо за умовою)
4) Cн = R * MEн CP: R (ігноруємо за умовою)
5) Xн = a * d * Zн + a* Cн CP: a, d, копія

Bн – H елементів вектора B, ( H = N/P )
N – розмір векторів, матриць
P – кількість процесорів (потоків) => P = 4
H = N/4

## ІІ етап. Розробка алгоритмів потоків

**Т1** **Точки синхронізації**

1) Введення МО
2) Сигнал Т2, Т3, Т4 про введення МО --$S_{2-1}$ $S_{3-1}$ $S_{4-1}$
3) Чекати на введення даних з Т2, Т3, Т4 --$W_{2-1}$ $W_{3-1}$ $W_{4-1}$
4) Обчислення a1 = Вн * Zн
5) Обчислення a = a + a1 КД1
6) Сигнал Т2, Т3, Т4 про завершення обчислення a --$S_{2-2}$ $S_{3-2}$ $S_{4-2}$
7) Чекати на завершення обчислень a з Т2, Т3, Т4 --$W_{2-2}$ $W_{3-2}$ $W_{4-2}$
8) Обчислення МЕн = MRн* МО
9) Сигнал Т2, Т3, Т4 про завершення обчислення МЕн
10) Чекати на завершення обчислень Т2, Т3, Т4
11) Обчислення Сн = R * МЕн
12) Сигнал Т2, Т3, Т4 про завершення обчислення Сн
13) Чекати на завершення обчислень Т2, Т3, Т4
14) Копіювання a1 = a КД2
15) Копіювання d1 = d КД3
16) Обчислення Хн = a1 * d1 * Zн + a1*Сн
17) Сигнал Т4 на завершення обчислень Хн --$S_{4-3}$

**Т2**

1) Введення Z, R
2) Сигнал Т1, Т3, Т4 про введення Z, R --$S_{1-1}$ $S_{3-1}$ $S_{4-1}$
3) Чекати на введення даних з Т1, Т3, Т4 --$W_{1-1}$ $W_{3-1}$ $W_{4-1}$
4) Обчислення a2 = Вн * Zн
5) Обчислення a = a + a2 КД1
6) Сигнал Т1, Т3, Т4 про завершення обчислення a --$S_{1-2}$ $S_{3-2}$ $S_{4-2}$
7) Чекати на завершення обчислень a з Т1, Т3, Т4 --$W_{1-2}$ $W_{3-2}$ $W_{4-2}$
8) Обчислення МЕн = MRн* МО
9) Сигнал Т1, Т3, Т4про завершення обчислення МЕн
10) Чекати на завершення обчислень Т1, Т3, Т4
11) Обчислення Сн = R * МЕн
12) Сигнал Т1, Т3, Т4про завершення обчислення Сн
13) Чекати на завершення обчислень Т1, Т3, Т4
14) Копіювання a2 = a КД2
15) Копіювання d2 = d КД3
16) Обчислення Хн = a2 * d2 * Zн + a2*Сн
17) Сигнал Т4 на завершення обчислень Хн --$S_{4-3}$

**Т3**

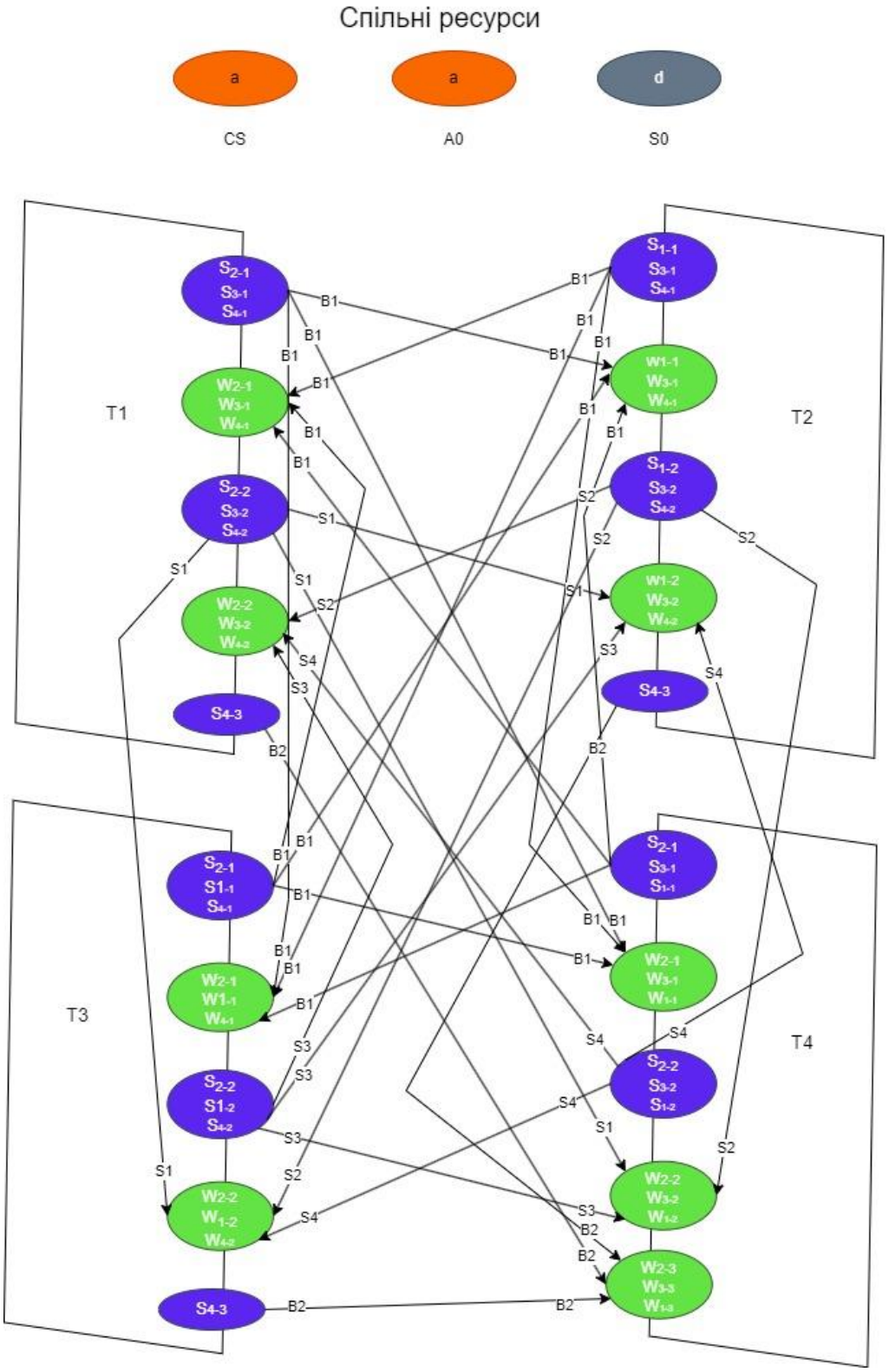
1) Введення B, MR
2) Сигнал T1, T2, T4 про введення B, MR --$S_{1-1}$ $S_{2-1}$ $S_{4-1}$
3) Чекати на введення даних з T1, T2, T4 --$W_{1-1}$ $W_{2-1}$ $W_{4-1}$
4) Обчислення a3 = Bн * Zн
5) Обчислення a = a + a3 КД1
6) Сигнал T1, T2, T4 про завершення обчислення a --$S_{1-2}$ $S_{2-2}$ $S_{4-2}$
7) Чекати на завершення обчислень a з T1, T2, T4 --$W_{1-2}$ $W_{2-2}$ $W_{4-2}$
8) Обчислення MEн = MRн* MO
9) Сигнал T1, T2, T4 про завершення обчислення MEн
10) Чекати на завершення обчислень T1, T2, T4
11) Обчислення Cн = R * MEн
12) Сигнал T1, T2, T4 про завершення обчислення Cн
13) Чекати на завершення обчислень T1, T2, T4
14) Копіювання a3 = a КД2
15) Копіювання d3 = d КД3
16) Обчислення Xн = a3 * d3 * Zн + a3*Cн
17) Сигнал T4 на завершення обчислень Xн --$S_{4-3}$

**Т4**

1) Введення d
2) Сигнал T1, T2, T3 про введення d --$S_{1-1}$ $S_{2-1}$ $S_{3-1}$
3) Чекати на введення даних з T1, T2, T3 --$W_{1-1}$ $W_{2-1}$ $W_{3-1}$
4) Обчислення a4 = Bн * Zн
5) Обчислення a = a + a4 КД1
6) Сигнал T1, T2, T3 про завершення обчислення a --$S_{1-2}$ $S_{2-2}$ $S_{3-2}$
7) Чекати на завершення обчислень a з T1, T2, T3 --$W_{1-2}$ $W_{2-2}$ $W_{3-2}$
8) Обчислення MEн = MRн* MO
9) Сигнал T1, T2, T3 про завершення обчислення MEн
10) Чекати на завершення обчислень T1, T2, T3
11) Обчислення Cн = R * MEн
12) Сигнал T1, T2, T3 про завершення обчислення Cн
13) Чекати на завершення обчислень T1, T2, T3
14) Копіювання a4 = a КД2
15) Копіювання d4 = d КД3
16) Обчислення Xн = a4 * d4 * Zн + a4*Cн
17) Чекати на завершення обчислень Xн в T1, T2, T3 --$W_{1-3}$ $W_{2-3}$ $W_{3-3}$
18) Виведення результату Xн

## III Етап. Розробка схеми взаємодії задач

При розробці схеми взаємодії задач, що ілюстровано на мал. 1, було враховано засоби організації синхронізації які присутні в мові Java, а саме **семафори**, **критичні секції**, **атомік-змінні** і **бар'єри**




Мал 1 схема взаємодії задач

**B1** – бар’єр для синхронізації потоків вводу T1, T2, T3, T4

**S1, S2, S3, S4 –** семафори про завершення обчислення a
**B2 –** бар’єр для синхронізації потоків про завершення обчислення X

**S0** – семафор для копіювання КД3, захист спільного ресурсу d
**A0** – атомік-змінна для КД1, захист спільного ресурсу a
**CS** – критична секція для КД2, захист спільного ресурсу a

## IV Етап розробки програми

На основі побудованої паралельного математичної алгоритму(1 етап), розробці його потоків(2 етап) та ілюстрування схеми взаємодії задач(3 етап) було реалізовано програму на мові Java

Програма складається з основного класу Lab2, допоміжного класу Data, що зберігає всі спільні змінні та містить методи для роботи над векторами і матрицями, а також класи Thread_1, Thread_2, Thread_3, Thread_4 для відповідних потоків T1, T2, T3, T4 в яких реалізований відповідний алгоритм обчислення

**Лістинг програми**

### Lab2.java

```
c
```

### Thread_1.java

```
package org.example;

import java.util.concurrent.BrokenBarrierException;

public class Thread_1 extends Thread {
    private Data data;
    final int id = 1;
    private int ai;
    private int di;

    public Thread_1(Data d) {
        data = d;
    }

    @Override
    public void run() {
        System.out.println("T1 is started!");

        try {
            data.MO =data.fillMatrix(data.MO);
            data.MR = data.fillMatrix(data.MR);
            data.B = data.fillVector(data.B);
            data.R = data.fillVector(data.R);
            data.Z = data.fillVector(data.Z);
```

```
                                                // сигнал про введення даних
+ очікування, на ввід інших потоків своїх даних
            data.B1.await();                    // бар'єр B1

            int indexStart = (id - 1) * data.H;
            int indexEnd =  id * data.H;

            // Обчислення a1 = Bн * Zн
            int []Bh = Data.getSubvector(data.B,indexStart, indexEnd);
            int []Zh = Data.getSubvector(data.B,indexStart, indexEnd);
            ai = data.multiplyVectors(Bh, Zh);

                                                                    // доступ
до спільного ресурсу - КД1
            data.a.updateAndGet(current -> current + ai);          // атомік-
змінна a
                                                // сигнал про завершення
обчислення a = a + a1
            data.S1.release(3);      // семафор S1

            data.S2.acquire();              // очікування на завершення
обчислень a з T2, T3, T4
            data.S3.acquire();              // Семафори S2, S3, S4
            data.S4.acquire();

            int [][] MRh = Data.getSubmatrix(data.MO, indexStart, indexEnd);

            int [][] MEh = Data.multiplyMatrices(data.MO, MRh, indexStart,
indexEnd); // Обчислення MEн = MOн* MR

            int []Ch = Data.multiplyVectorByMatrix(data.R, MEh);
// Обчислення Cн = R * MEн

                                          // копіювання ai = a --- КД2
            ai = data.CS1();             // критична секція CS1

                                         // копіювання di = d --- КД3
            data.S0.acquire();          // семафор S0
            di = data.d;
            data.S0.release();

            int[] Xh = Data.calculateXh(Ch, Zh, ai, di);
//Обчислення Xн = a1 * d1 * Zн + a1*Cн

            for (int i = indexStart; i < indexEnd; i++) {
                data.X[i] += Xh[i - indexStart];
            }
                                        // Сигнал про завершення обчислення X
            data.B2.await();            // Бар'єр B2


        } catch (InterruptedException | BrokenBarrierException e) {
            throw new RuntimeException(e);
        } finally {
            System.out.println("T1 is finished!");
        }
    }
}
```

**Thread_2.java**

```java
package org.example;

import java.util.Arrays;
import java.util.concurrent.BrokenBarrierException;

public class Thread_2 extends Thread {
    private Data data;
    final int id = 2;
    private int ai;
    private int di;

    public Thread_2(Data d) {
        data = d;
    }

    @Override
    public void run() {
        System.out.println("T2 is started!");

        try {
            data.MO =data.fillMatrix(data.MO);
            data.MR = data.fillMatrix(data.MR);
            data.B = data.fillVector(data.B);
            data.R = data.fillVector(data.R);
            data.Z = data.fillVector(data.Z);
                                                // сигнал про введення даних
+ очікування, на ввід інших потоків своїх даних
            data.B1.await();                    // бар'єр B1

            int indexStart = (id - 1) * data.H;
            int indexEnd =  id * data.H;

            // Обчислення a2 = Bн * Zн
            int []Bh = Data.getSubvector(data.B,indexStart, indexEnd);
            int []Zh = Data.getSubvector(data.B,indexStart, indexEnd);
            ai = data.multiplyVectors(Bh, Zh);

                                                                    // доступ
до спільного ресурсу - КД1
            data.a.updateAndGet(current -> current + ai);          // атомік-
змінна a
            // сигнал про завершення обчислення a = a + a2
            data.S2.release(3);      // семафор S2

            data.S1.acquire();              // очікування на завершення
обчислень a з T1, T3, T4
            data.S3.acquire();              // Семафори S1, S3, S4
            data.S4.acquire();

            int [][] MRh = Data.getSubmatrix(data.MO, indexStart, indexEnd);

            int [][] MEh = Data.multiplyMatrices(data.MO, MRh, indexStart,
indexEnd); // Обчислення MEн = MOн* MR

            int []Ch = Data.multiplyVectorByMatrix(data.R, MEh);
// Обчислення Cн = R * MEн

                                        // копіювання ai = a --- КД2
            ai = data.CS1();            // критична секція CS1

                                       // копіювання di = d --- КД3
            data.S0.acquire();         // семафор S0
            di = data.d;
            data.S0.release();
```

```
            int[] Xh = Data.calculateXh(Ch, Zh, ai, di);
//Обчислення Xн = a2 * d1 * Zн + a2*Cн

            for (int i = indexStart; i < indexEnd; i++) {
                data.X[i] += Xh[i - indexStart];
            }
                                        // Сигнал про завершення обчислення X
            data.B2.await();            // Бар'єр B2


        } catch (InterruptedException | BrokenBarrierException e) {
            throw new RuntimeException(e);
        } finally {
            System.out.println("T2 is finished!");
        }
    }
}
```

## Thread_3.java

```
package org.example;

import java.util.Arrays;
import java.util.concurrent.BrokenBarrierException;

public class Thread_3 extends Thread {
    private Data data;
    private int ai;
    private int di;
    final int id = 3;
    public Thread_3( Data d) {
        data = d;
    }

    @Override
    public void run() {

        System.out.println("T3 is started!");

        try {
            data.MO =data.fillMatrix(data.MO);
            data.MR = data.fillMatrix(data.MR);
            data.B = data.fillVector(data.B);
            data.R = data.fillVector(data.R);
            data.Z = data.fillVector(data.Z);
                                            // сигнал про введення даних +
очікування, на ввід інших потоків своїх даних
            data.B1.await();                // бар'єр B1

            int indexStart = (id - 1) * data.H;
            int indexEnd =  id * data.H;

            // Обчислення a3 = Bн * Zн
            int []Bh = Data.getSubvector(data.B,indexStart, indexEnd);
            int []Zh = Data.getSubvector(data.B,indexStart, indexEnd);
            ai = data.multiplyVectors(Bh, Zh);
                                                                    // доступ
до спільного ресурсу - КД1
            data.a.updateAndGet(current -> current + ai);          // атомік-
змінна a
                                            // сигнал про завершення
```

```
обчислення a = a + a3
            data.S3.release(3);       // семафор S3

            data.S1.acquire();              // очікування на завершення
обчислень a з T1, T2, T4
            data.S2.acquire();              // Семафори S1, S2, S4
            data.S4.acquire();

            int [][] MRh = Data.getSubmatrix(data.MO, indexStart, indexEnd);

            int [][] MEh = Data.multiplyMatrices(data.MO, MRh, indexStart,
indexEnd); // Обчислення МЕн = МОн* MR

            int []Ch = Data.multiplyVectorByMatrix(data.R, MEh);
// Обчислення Сн = R * МЕн

                                          // копіювання ai = a --- КД2
            ai = data.CS1();             // критична секція CS1

                                          // копіювання di = d --- КД3
            data.S0.acquire();           // семафор S0
            di = data.d;
            data.S0.release();

            int[] Xh = Data.calculateXh(Ch, Zh, ai, di);
//Обчислення Хн = a3 * d1 * Zн + a3*Сн

            for (int i = indexStart; i < indexEnd; i++) {
                data.X[i] += Xh[i - indexStart];
            }
                                         // Сигнал про завершення обчислення X
            data.B2.await();             // Бар'єр B2

        } catch (InterruptedException | BrokenBarrierException e) {
            throw new RuntimeException(e);
        }
        finally {
            System.out.println("T3 finished");
        }

    }
}
```

## Thread_4.java

```
package org.example;

import java.util.Arrays;
import java.util.concurrent.BrokenBarrierException;

public class Thread_4 extends Thread {
    final Data data;
    private int ai;
    private int di;
    final int id = 4;
    public Thread_4( Data d) {

        data = d;
    }

    @Override
    public void run() {
```

```
        System.out.println("T4 is started!");

        try {
            data.MO =data.fillMatrix(data.MO);
            data.MR = data.fillMatrix(data.MR);
            data.B = data.fillVector(data.B);
            data.R = data.fillVector(data.R);
            data.Z = data.fillVector(data.Z);
                                                // сигнал про введення даних
+ очікування, на ввід інших потоків своїх даних
            data.B1.await();                    // бар'єр B1

            int indexStart = (id - 1) * data.H;
            int indexEnd =  id * data.H;

            // Обчислення a4 = Bн * Zн
            int []Bh = Data.getSubvector(data.B,indexStart, indexEnd);
            int []Zh = Data.getSubvector(data.B,indexStart, indexEnd);
            ai = data.multiplyVectors(Bh, Zh);

                                                                    // доступ
до спільного ресурсу - КД1
            data.a.updateAndGet(current -> current + ai);          // атомік-
змінна a
                                            // сигнал про завершення
обчислення a = a + a4
            data.S4.release(3);      // семафор S4

            data.S1.acquire();              // очікування на завершення
обчислень a з T1, T2, T3
            data.S2.acquire();              // Семафори S1, S2, S3
            data.S3.acquire();

            int [][] MRh = Data.getSubmatrix(data.MO, indexStart, indexEnd);

            int [][] MEh = Data.multiplyMatrices(data.MO, MRh, indexStart,
indexEnd); // Обчислення MEн = MOн* MR

            int []Ch = Data.multiplyVectorByMatrix(data.R, MEh);
// Обчислення Cн = R * MEн

                                        // копіювання ai = a --- КД2
            ai = data.CS1();            // критична секція CS1

                                       // копіювання di = d --- КД3
            data.S0.acquire();         // семафор S0
            di = data.d;
            data.S0.release();

            int[] Xh = Data.calculateXh(Ch, Zh, ai, di);
//Обчислення Xн = a4 * d1 * Zн + a4*Cн

            for (int i = indexStart; i < indexEnd; i++) {
                data.X[i] += Xh[i - indexStart];
            }
                                       // Сигнал про завершення обчислення X
            data.B2.await();           // Бар'єр B2

            System.out.println(Arrays.toString(data.X));             //
Виведення результату

        } catch (InterruptedException | BrokenBarrierException e) {
            throw new RuntimeException(e);
```

```
        }
        finally {
            System.out.println("T4 finished");
        }

    }
}
```

## Data.java

```
package org.example;

import java.util.Arrays;
import java.util.concurrent.CyclicBarrier;
import java.util.concurrent.Semaphore;
import java.util.concurrent.atomic.AtomicInteger;

public class Data {

    public int N = 12;
    public int P = 4;
    public int H = N / P;
    public int[][] MO = new int[N][N];
    public int[][] MR = new int[N][N];
    public int[] B = new int[N];
    public int[] Z = new int[N];
    public int[] R = new int[N];
    public int[] X = new int[N];
    public int d = 1;

    Semaphore S0 = new Semaphore(1);
    Semaphore S1 = new Semaphore(0);
    Semaphore S2 = new Semaphore(0);
    Semaphore S3 = new Semaphore(0);
    Semaphore S4 = new Semaphore(0);
    CyclicBarrier B1 = new CyclicBarrier(4);
    CyclicBarrier B2 = new CyclicBarrier(4);
    public AtomicInteger a = new AtomicInteger(0);

    public synchronized int CS1() {
        return a.intValue();
    }

    public int [] fillVector(int[] vector) {
        for (int i = 0; i < vector.length; i++) {
            vector[i] = 1;
        }
        return vector;
    }

    public int[][] fillMatrix(int[][] matrix) {
        for (int i = 0; i < N; i++) {
            for (int j = 0; j < N; j++) {
                matrix[i][j] = 1;
            }
        }
        return matrix;
    }

    public static int[][] getSubmatrix(int[][] matrix, int start, int end) {
        if (start < 0 || start >= matrix[0].length || end <= start || end >
matrix[0].length) {
```

```
            throw new IllegalArgumentException("Invalid column indices");
        }

        int rows = matrix.length;
        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 start, int end) {
        if (start < 0 || start >= vector.length || end <= start || end >
vector.length) {
            throw new IllegalArgumentException("Invalid indices");
        }

        int size = end - start;
        int[] subvector = new int[size];

        for (int i = start; i < end; i++) {
            subvector[i - start] = vector[i];
        }

        return subvector;
    }

    public static int multiplyVectors(int[] vector1, int[] vector2) {
        if (vector1.length != vector2.length) {
            throw new IllegalArgumentException("Vectors must have the same
length");
        }

        int result = 0;

        for (int i = 0; i < vector1.length; i++) {
            result += vector1[i] * vector2[i];
        }

        return result;
    }

    public static int[][] multiplyMatrices(int[][] matrix1, int[][] matrix2,
int start, int end) {
        int rowsMatrix1 = matrix1.length;

        int[][] result = new int[rowsMatrix1][end - start];

        for (int i = 0; i < rowsMatrix1; i++) {
            for (int j = 0; j < end-start; j++) {
                for (int k = 0; k < rowsMatrix1; k++) {
                    result[i][j] += matrix1[i][k] * matrix2[k][j];
                }
            }
        }

        return result;
    }
```

```java
    public static int[] multiplyVectorByMatrix(int[] vector, int[][] matrix)
{

        int[] result = new int[matrix.length];

        for (int i = 0; i < matrix[0].length; i++) {
            for (int j = 0; j < matrix.length; j++) {
                result[i] += matrix[j][i] * vector[j];
            }

        }
        return result;
    }

    public static int[] calculateXh(int[] C, int[] Z, int a, int d) {
        int length = Math.min(C.length, Z.length);
        int[] result = new int[length];

        for (int i = 0; i < length; i++) {
            result[i] = a * d * Z[i] + a * C[i];
        }

        return result;
    }
}
```

## Результат виконання програми

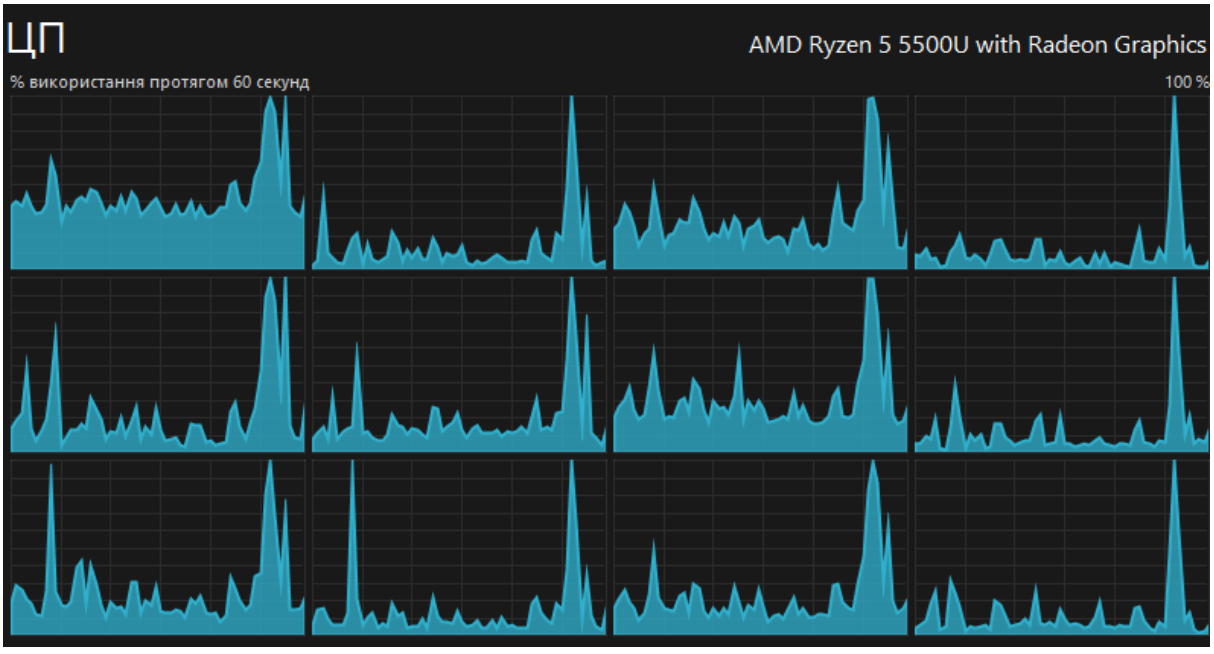
```
C:\Users\user\.jdks\openjdk-21.0.1\bin\java.exe "-javaagent:D:\api\Intelli
T2 is started!
T3 is started!
T1 is started!
T4 is started!
T3 finished
T1 is finished!
T2 is finished!
[1740, 1740, 1740, 1740, 1740, 1740, 1740, 1740, 1740, 1740, 1740, 1740]
T4 finished
```

Результат виконання програми при N=12

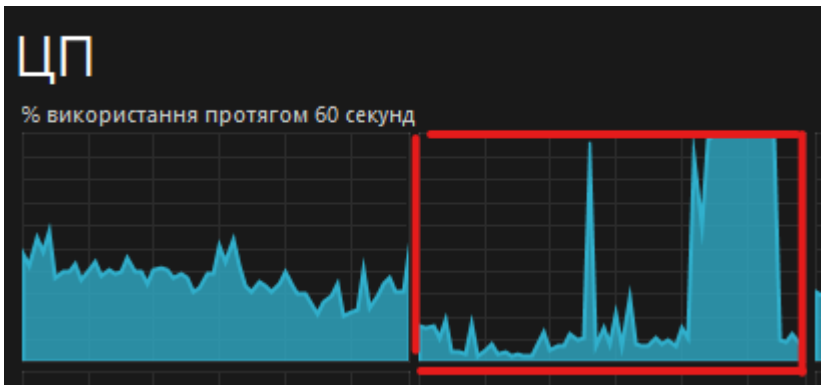
## Тестування

Опис комп'ютера:
*Процесор:* AMD Ryzen 5 5500U
*Кількість ядер:* 6
*Кількість логічних ядер:* 12

Для N=1000, було проаналізовано процес завантаження програмою ядер багатоядерного процесора за допомогою **Диспетчеру задач ОС Windows.**




Мал. 2 Навантаження при автоматичному розподілі ядер




Мал 3. Навантаження на 1 ядро

Визначення часу виконання програми при:

- Автоматичному розподілені навантаження на ядра ~ **54.6**
- При навантажені на одне ядро ~ **73.2**

Кп = **73.2/ 54.6 ~ 1,34**

# ВИСНОВОК

1) Виконано розробку програми з використанням засобів мови Java:
   - Потоки (Threads)
   - клас concurrent для використання семафорів, бар’єрів та атомік-змінних
   - Обробка виключень (Exception Handling)
2) У процесі розробки було розраховано паралельний математичний алгоритм, який дає змогу виконати обчислення заданої формули та визначити спільні ресурси
3) Був розроблений алгоритм потоків, кожен з яких виконує свою задачу(розрахування формули). На його основі визначено спільні ресурси(скаляри a, d) і відповідні операції копіювання та перезапису. Окремо визначені відповідні точки синхронізації – введення даних, виконання обчислень, виведення. Визначені критичні ділянки.
4) Побудовано схему взаємодії задач та визначені наступні засоби захисту та взаємодії: семафори(сигнали про початок та закінчення обчислень), бар’єри(введення виведення даних), атомік-змінні(атомарність перезапису) та критичні секції(безпечне копіювання).
5) Проведено тестування програми з метою визначення  завантаження ядер процесора. Значення коефіцієнта прискорення дорівнює 1,34