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

з дисципліни
«Програмне забезпечення високопродуктивних комп’ютерних систем»

на тему

«Семафори, критичні секції, атомік-змінні, бар’єри»

Виконав
Студент групи ІМ-13
Котенко Ярослав Олегович

Перевірив
доц. Корочкін О.В.

Київ 2024

Завдання

Розробити паралельну програму для обчислення в паралельної системі (ПКС СП) функції :

Варіант: 22

W = max(C*MD)*C + E*(MA*MB)*d

Введення – виведення даних			
1	2	3	4
-	MB, E	C, MA, W	MD, d

Мова програмування: Java

Засоби організації взаємодії процесів: семафори,критичні секції, атомік змінні (типи), бар'єри.

Виконання лабораторної роботи

Етап 1. Побудова паралельного математичного алгоритму.

W = max(C*MD)*C + E*(MA*MB)*d

1) Ан = C * MDн CP: C
2) ai = max(Ан) i = 1..4
3) a = max(a,ai) CP: a
4) MC = MA * MBн CP: MA
5) Хн = E * MCн CP: E
6) Wн = a * Cн + d * Хн CP: a, d - копії.

N - розмірність вектора/матриці.

P - кількість потоків, які виконують обчислення.

H = N / P

Етап 2. Розробка алгоритмів потоків.

Задача T1 **Точки синхронізації**

1. **Чекати** на введення даних в задачах T2,T3,T4 **-- W2-1, W3-1,W4-1**
2. Обчислення 1: Ан = C * MDн

3. Обчислення 2: ai = max(Ан)
4. Обчислення 3: a = max(a, ai) – КД1
5. **Сигнал** задачам T2,T3,T4 про завершення обчислення 3 **-- S2-1,S3-1,S4-1**
6. **Чекати** на завершення обчислення 3 в задачах T2,T3,T4 **– W2-2,W3-2,W4-2**
7. Обчислення 4: MCн = MA * MBн
8. Обчислення 5: Xн = E * MCн
9. **Копія** a1 = a – КД2
10. **Копія** d1 = d – КД3
11. Обчислення 6: Wн = a1 * Cн + d1 * Xн
12. **Чекати** завершення обчислення 6 в задачах T2,T3,T4 **– W2-3,W3-3,W4-3**
13. Виведення результату W

Задача T2

1. Введення MB, E
2. **Сигнал** задачам T1,T3,T4 про введення MB, E **– S1-1. S3-1, S4-1**
3. **Чекати** на введення даних в задачах T3,T4 **– W3-1,W4-1**
4. Обчислення 1: Ан = C * MDн
5. Обчислення 2: ai = max(Ан)
6. Обчислення 3: a = max(a, ai) – КД1
7. **Сигнал** задачам T1,T3,T4 про завершення обчислення 3 **– S1-2,S3-2,S4-2**
8. **Чекати** на завершення обчислення 3 в задачах T1,T3,T4 **– W1-2,W3-2,W4-2**
9. Обчислення 4: MCн = MA * MBн
10. Обчислення 5: Xн = E * MCн
11. **Копія** a2 = a – КД2
12. **Копія** d2 = d – КД3
13. Обчислення 6: Wн = a2 * Cн + d2 * Xн
14. **Сигнал** задачі T1 про завершення обчислення 6 **– S1-3**

Задача T3

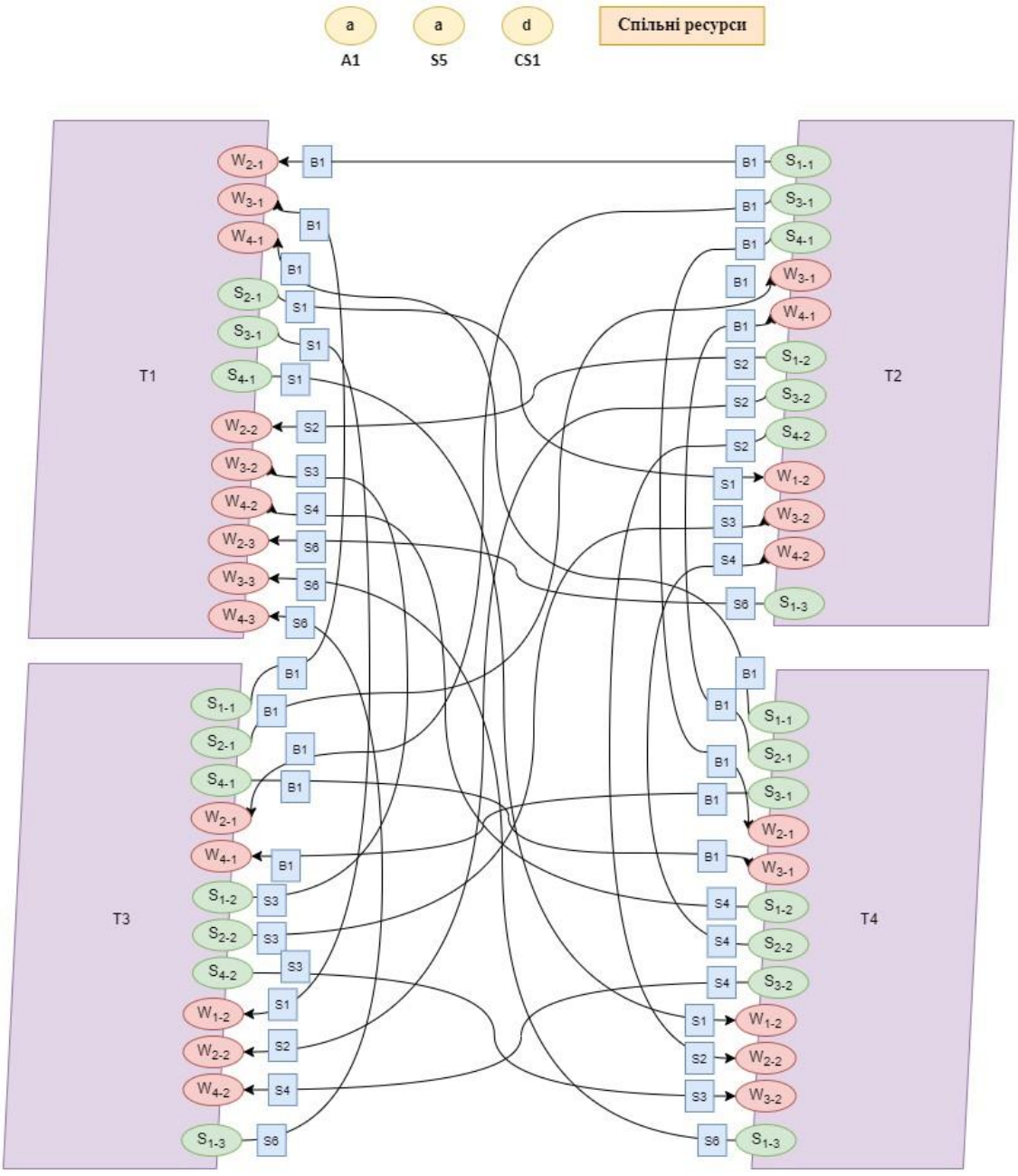
1. Введення C, MA, W
2. **Сигнал** задачам T1,T2,T4 про введення C, MA, W **– S1-1,S2-1,S4-1**
3. **Чекати** на введення даних в задачах T2,T4 **– W2-1,W4-1**
4. Обчислення 1: Ан = C * MDн
5. Обчислення 2: ai = max(Ан)
6. Обчислення 3: a = max(a, ai) – КД1
7. **Сигнал** задачам T1,T2,T4 про завершення обчислення 3 **– S1-2,S2-2,S4-2**
8. **Чекати** на завершення обчислення 3 в задачах T1,T2,T4 **– W1-2,W2-2,W4-2**
9. Обчислення 4: MCн = MA * MBн
10. Обчислення 5: Xн = E * MCн
11. **Копія** a3 = a – КД2
12. **Копія** d3 = d – КД3
13. Обчислення 6: Wн = a3 * Cн + d3 * Xн
14. **Сигнал** задачі T1 про завершення обчислення 6 **– S1-3**

Задача T4

1. Введення MD, d

2. **Сигнал** задачам T1,T2,T3 про введення MD, d **– S1-1,S2-1,S3-1**
3. **Чекати** на введення даних в задачах T2,T3 **– W2-1,W3-1**
4. Обчислення 1: Ан = C * MDн
5. Обчислення 2: ai = max(Ан)
6. Обчислення 3: a = max(a, ai) – КД1
7. **Сигнал** задачам T1,T2,T3 про завершення обчислення 3 **– S1-2,S2-2,S3-2**
8. **Чекати** на завершення обчислення 3 в задачах T1,T2,T3 **– W1-2,W2-2,W3-2**
9. Обчислення 4: МСн = МА * МВн
10. Обчислення 5: Хн = E * МСн
11. **Копія** a4 = a – КД2
12. **Копія** d4 = d – КД3
13. Обчислення 6: Wн = a4 * Сн + d4 * Хн
14. **Сигнал** задачі T1 про завершення обчислення 6 **– S1-3**

Етап 3. Розробка схеми взаємодії задач



Кожна задача має три критичні ділянки (КД), і кожна з них обробляється за допомогою різних засобів: КД1 використовує атоміки, КД2 - семафори, а КД3 - критичну секцію.

Семафори:

S1, S2, S3, S4 - семафори для сигналів про завершення обчислення 3.

S5 - семафор для захисту КД2 (копіювання спільного ресурсу **a**).

S6 - семафор для сигналів до T1 про завершення задачами обчислення 6.

Бар'єр:
B1 - бар'єр для синхронізації введення даних з потоків T2, T3, T4.

Атомік-змінна:
a - для захисту КД1 (перезапис спільного ресурсу).

Критична секція:
Критична секція CS1 для захисту КД3 (копіювання спільного ресурсу **d**).

Етап 4. Розроблення програми.

Програма складається з наступних компонентів:
основний клас **Main**, який є вхідною точкою програми;
клас **Data**, що містить усі загальні змінні та статичні методи для роботи з векторами та матрицями;
а також класи **T1**, **T2**, **T3**, **T4**, які відповідають відповідним потокам у програмі.

Main:

```
// Програмне забезпечення високопродуктивних комп'ютерних систем
// Лабораторна робота №2
// Варіант 22
// W = max(C*MD)*C + E*(MA*MB)*d
// Котенко Ярослав Олегович
// Група ІМ-13
// 29.03.2024

import java.text.DecimalFormat;

public class Main {
    public static void main(String[] args) throws InterruptedException {
        Data data = new Data();

        long startTime = System.nanoTime();

        T1 t1 = new T1(data);
        T2 t2 = new T2(data);
        T3 t3 = new T3(data);
        T4 t4 = new T4(data);
```

```
        t1.start();
        t2.start();
        t3.start();
        t4.start();

        t1.join();
        t2.join();
        t3.join();
        t4.join();

        long endTime = System.nanoTime();

        double duration = (double) (endTime - startTime) / 1_000_000;

        DecimalFormat df = new DecimalFormat("0.00");
        String formattedDuration = df.format(duration);

        System.out.println("Time " + formattedDuration + " ms");

    }
}
```

Data:

```
import java.util.Arrays;
import java.util.concurrent.CyclicBarrier;
import java.util.concurrent.Semaphore;
import java.util.concurrent.atomic.AtomicInteger;
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;

public class Data {
    public int N = 16;
    public int P = 4;
    public int H = N / P;
    AtomicInteger a = new AtomicInteger(0);
    int d;
    int[] W = new int[N];
    int[] C = new int[N];
    int[] E = new int[N];
    int[] A = new int[N];
    int[][] MD = new int[N][N];
    int[][] MA = new int[N][N];
    int[][] MB = new int[N][N];
    int[][] MC = new int[N][N];

    CyclicBarrier barrier1 = new CyclicBarrier(4);

    Semaphore semaphore_1 = new Semaphore(0);
    Semaphore semaphore_2 = new Semaphore(0);
    Semaphore semaphore_3 = new Semaphore(0);
    Semaphore semaphore_4 = new Semaphore(0);
    Semaphore semaphore_5 = new Semaphore(1);
    Semaphore semaphore_6 = new Semaphore(0);


    static Lock pointLock = new ReentrantLock();

    // Метод для обчислення 1: Ан = С * MDн
    public static int[] calculateStep1(int[] C, int[][] MD_H) {
        return multiplyVectorByMatrix(C, MD_H);
```

```java
    }
    // Метод для обчислення 4: МСн = МА * МВн
    public static int[][] calculateStep4(int[][] MA, int[][] MB_H) {
        return Data.multiplyMatrices(MA, MB_H);
    }
    // Метод для обчислення 5: Хн = Е * МСн
    public static int[] calculateStep5(int[] E, int[][] MC_H) {
        return multiplyVectorByMatrix(E, MC_H);
    }
    // Метод для обчислення 6: Wн = a4 * Сн + d4 * Хн
    public static int[] calculateStep6(int a, int d, int[] C_H, int[] X_H) {
        return addVectors(scalarMultiply(a, C_H), scalarMultiply(d, X_H));
    }

    // Метод для копіювання d використовуючи lock
    public static int copyD_CS(Data data) {
        pointLock.lock();
        try {
            return data.d;
        } finally {
            pointLock.unlock();
        }
    }

    // Метод для множення скаляру на вектор
    public static int[] scalarMultiply(int scalar, int[] vector) {
        int[] result = new int[vector.length];
        for (int i = 0; i < vector.length; i++) {
            result[i] = scalar * vector[i];
        }
        return result;
    }
    // Метод для додавання двох векторів
    public static int[] addVectors(int[] vectorA, int[] vectorB) {
        int[] result = new int[vectorA.length];
        for (int i = 0; i < vectorA.length; i++) {
            result[i] = vectorA[i] + vectorB[i];
        }
        return result;
    }

    // Метод для множення двох матриць
    public static int[][] multiplyMatrices(int[][] matrix1, int[][] matrix2)
{
        int rows1 = matrix1.length;
        int columns1 = matrix1[0].length;
        int columns2 = matrix2[0].length;

        int[][] result = new int[rows1][columns2];

        for (int i = 0; i < rows1; i++) {
            for (int j = 0; j < columns2; j++) {
                for (int k = 0; k < columns1; k++) {
                    result[i][j] += matrix1[i][k] * matrix2[k][j];
                }
            }
        }

        return result;
    }

    // Метод для знаходження максимуму в векторі
    public static int findMaxInVector(int[] vector) {
        return Arrays.stream(vector).max().getAsInt();
```

```
    }

    // Метод для множення вектора на матрицю
    public static int[] multiplyVectorByMatrix(int[] vector, int[][] matrix)
{
        int vectorLength = vector.length;
        int matrixColumns = matrix[0].length;

        int[] result = new int[matrixColumns];

        for (int i = 0; i < matrixColumns; i++) {
            int sum = 0;
            for (int j = 0; j < vectorLength; j++) {
                sum += vector[j] * matrix[j][i];
            }
            result[i] = sum;
        }

        return result;
    }

    // Метод для отримання підматриці з стовпців, за заданими координатами
початку та кінця
    public static int[][] getSubmatrixFromColumns(int[][] sourceMatrix, int
startColumn, int endColumn) {
        int rowCount = sourceMatrix.length;
        int columnCount = endColumn - startColumn;
        int[][] submatrix = new int[rowCount][columnCount];
        for (int i = 0; i < rowCount; i++) {
            for (int j = startColumn; j < endColumn; j++) {
                submatrix[i][j - startColumn] = sourceMatrix[i][j];
            }
        }
        return submatrix;
    }

    // Метод для отримання підвектора, за заданими координатами початку та
кінця
    public static int[] getSubvector(int[] sourceVector, int startPos, int
endPos) {
        int size = endPos - startPos;
        int[] subvector = new int[size];

        for (int i = startPos; i < endPos; i++) {
            subvector[i - startPos] = sourceVector[i];
        }
        return subvector;
    }

    // Метод для об'єднання частинних результатів в один - фінальний
    public static void insertIntoFullW(Data data, int[] sub_W, int startPos,
int endPos) {
        for (int i = startPos, j = 0; i < endPos; i++, j++) {
            data.W[i] = sub_W[j];
        }
    }

    // Виведення фінально результату
    public static void printW(Data data) {
        System.out.println(Arrays.toString(data.W));
    }
}
```

T1:

```
public class T1 extends Thread {
    int Tid = 1;
    Data data;
    int startPos = 0;
    int endPos;
    public T1(Data data) {
        this.data = data;
        endPos = Tid * data.H;
    }

    @Override
    public void run() {
        System.out.println("T1 start");
        try {
            // B1 - бар'єр, що сигналізує про очікування введення даних усіма
іншими потоками.
            data.barrier1.await();

            // Обчислення 1
            int[][] MD_H = Data.getSubmatrixFromColumns(data.MD, startPos,
endPos);
            int[] A_H = Data.calculateStep1(data.C, MD_H);

            // Обчислення 2
            int ai = Data.findMaxInVector(A_H);

            // Обчислення 3
            int finalAi = ai;
            // КД 1 . Атомік змінна
            data.a.updateAndGet(a -> Math.max(a, finalAi));

            // S1 - сигнал про завершення обчислення 3
            data.semaphore_1.release(3);

            // S2,S3,S4. Очікування, поки всі потоки завершать обчислення 3
            data.semaphore_2.acquire();
            data.semaphore_3.acquire();
            data.semaphore_4.acquire();

            // Обчислення 4
            int[][] MB_H = Data.getSubmatrixFromColumns(data.MB, startPos,
endPos);
            int[][] MC_H = Data.calculateStep4(data.MA, MB_H);

            // Обчислення 5
            int[] X_H = Data.calculateStep5(data.E, MC_H);

            // КД 2. S5 - семафор. Копія a1
            data.semaphore_5.acquire();
            ai = Integer.parseInt(String.valueOf(data.a));
            data.semaphore_5.release();

            // КД 3. Критична секція. Копія d1
            int di = Data.copyD_CS(data);

            // Обчислення 6
            int[] C_H = Data.getSubvector(data.C, startPos, endPos);

            int[] W_H = Data.calculateStep6(ai, di, C_H, X_H);
```

```
            // Об'єднання фінального результату
            Data.insertIntoFullW(data, W_H, startPos, endPos);

            // чекати, щоб інші потоки завершили обчислення 6
            data.semaphore_6.acquire(3);

            // Виведення результату
            Data.printW(data);
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            System.out.println("T1 finish");
        }
    }
}
```

T2:

```
import java.util.Arrays;

public class T2 extends Thread {
    int Tid = 2;
    Data data;
    int startPos;
    int endPos;
    public T2(Data data) {
        this.data = data;
        this.startPos = (Tid - 1) * data.H;
        this.endPos = Tid * data.H;
    }

    @Override
    public void run() {
        System.out.println("T2 start");
        try {
            // Введення E, MB
            initData();
            // B1 - бар'єр, що сигналізує про очікування введення даних усіма
іншими потоками.
            data.barrier1.await();

            // Обчислення 1
            int[][] MD_H = Data.getSubmatrixFromColumns(data.MD, startPos,
endPos);
            int[] A_H = Data.calculateStep1(data.C, MD_H);

            // Обчислення 2
            int ai = Data.findMaxInVector(A_H);

            // Обчислення 3
            int finalAi = ai;
            // КД 1 . Атомік змінна
            data.a.updateAndGet(a -> Math.max(a, finalAi));

            // S2 - сигнал про завершення обчислення 3
            data.semaphore_2.release(3);

            // S1,S3,S4. Очікування, поки всі потоки завершать обчислення 3
            data.semaphore_1.acquire();
            data.semaphore_3.acquire();
            data.semaphore_4.acquire();
```

```
            // Обчислення 4
            int[][] MB_H = Data.getSubmatrixFromColumns(data.MB, startPos,
endPos);
            int[][] MC_H = Data.calculateStep4(data.MA, MB_H);

            // Обчислення 5
            int[] X_H = Data.calculateStep5(data.E, MC_H);

            // КД 2. S5 - семафор. Копія a2
            data.semaphore_5.acquire();
            ai = Integer.parseInt(String.valueOf(data.a));
            data.semaphore_5.release();

            // КД 3. Критична секція. Копія d2
            int di = Data.copyD_CS(data);

            // Обчислення 6
            int[] C_H = Data.getSubvector(data.C, startPos, endPos);

            int[] W_H = Data.calculateStep6(ai, di, C_H, X_H);

            // Об'єднання фінального результату
            Data.insertIntoFullW(data, W_H, startPos, endPos);

            // S6 - сигнал про завершення обчислення 6
            data.semaphore_6.release();

        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            System.out.println("T2 finish");
        }
    }

    public void initData() {
        Arrays.fill(data.E, 1);
        for(int i = 0; i < data.MB[0].length; i++) {
            Arrays.fill(data.MB[i], 1);
        }
    }
}
```

T3:

```
import java.util.Arrays;

public class T3 extends Thread {
    int Tid = 3;
    Data data;
    int startPos;
    int endPos;
    public T3(Data data) {
        this.data = data;
        this.startPos = (Tid - 1) * data.H;
        this.endPos = Tid * data.H;
    }

    @Override
    public void run() {
        System.out.println("T3 start");
        try {
```

```
            // Введення C, W, MA
            initData();
            // B1 - бар'єр, що сигналізує про очікування введення даних усіма
іншими потоками.
            data.barrier1.await();

            // Обчислення 1
            int[][] MD_H = Data.getSubmatrixFromColumns(data.MD, startPos,
endPos);
            int[] A_H = Data.calculateStep1(data.C, MD_H);

            // Обчислення 2
            int ai = Data.findMaxInVector(A_H);

            // Обчислення 3
            int finalAi = ai;
            // КД 1 . Атомік змінна
            data.a.updateAndGet(a -> Math.max(a, finalAi));

            // S3 - сигнал про завершення обчислення 3
            data.semaphore_3.release(3);

            // S1,S2,S4. Очікування, поки всі потоки завершать обчислення 3
            data.semaphore_1.acquire();
            data.semaphore_2.acquire();
            data.semaphore_4.acquire();

            // Обчислення 4
            int[][] MB_H = Data.getSubmatrixFromColumns(data.MB, startPos,
endPos);
            int[][] MC_H = Data.calculateStep4(data.MA, MB_H);

            // Обчислення 5
            int[] X_H = Data.calculateStep5(data.E, MC_H);

            // КД 2. S5 - семафор. Копія a3
            data.semaphore_5.acquire();
            ai = Integer.parseInt(String.valueOf(data.a));
            data.semaphore_5.release();

            // КД 3. Критична секція. Копія d3
            int di = Data.copyD_CS(data);

            // Обчислення 6
            int[] C_H = Data.getSubvector(data.C, startPos, endPos);
            int[] W_H = Data.calculateStep6(ai, di, C_H, X_H);

            // Об'єднання фінального результату
            Data.insertIntoFullW(data, W_H, startPos, endPos);

            // S6 - сигнал про завершення обчислення 6
            data.semaphore_6.release();
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            System.out.println("T3 finish");
        }
    }

    public void initData() {
        Arrays.fill(data.C, 1);
        Arrays.fill(data.W, 1);
        for(int i = 0; i < data.MA[0].length; i++) {
            Arrays.fill(data.MA[i], 1);
```

```
        }
    }
}
```

T4:

```
import java.util.Arrays;

public class T4 extends Thread {
    int Tid = 4;
    Data data;
    int startPos;
    int endPos;
    public T4(Data data) {
        this.data = data;
        this.startPos = (Tid - 1) * data.H;
        this.endPos = Tid * data.H;
    }

    @Override
    public void run() {
        System.out.println("T4 start");
        try {
            // Введення d, MD
            initData();
            // B1 - бар'єр, що сигналізує про очікування введення даних усіма
іншими потоками.
            data.barrier1.await();

            // Обчислення 1
            int[][] MD_H = Data.getSubmatrixFromColumns(data.MD, startPos,
endPos);
            int[] A_H = Data.calculateStep1(data.C, MD_H);

            // Обчислення 2
            int ai = Data.findMaxInVector(A_H);

            // Обчислення 3
            int finalAi = ai;
            // КД 1 . Атомік змінна
            data.a.updateAndGet(a -> Math.max(a, finalAi));

            // S4 - сигнал про завершення обчислення 3
            data.semaphore_4.release(3);

            // S1,S2,S3. Очікування, поки всі потоки завершать обчислення 3
            data.semaphore_1.acquire();
            data.semaphore_2.acquire();
            data.semaphore_3.acquire();

            // Обчислення 4
            int[][] MB_H = Data.getSubmatrixFromColumns(data.MB, startPos,
endPos);
            int[][] MC_H = Data.calculateStep4(data.MA, MB_H);

            // Обчислення 5
            int[] X_H = Data.calculateStep5(data.E, MC_H);

            // КД 2. S5 - семафор. Копія a4
            data.semaphore_5.acquire();
            ai = Integer.parseInt(String.valueOf(data.a));
            data.semaphore_5.release();

            // КД 3. Критична секція. Копія d4
            int di = Data.copyD_CS(data);
```

```
            // Обчислення 6
            int[] C_H = Data.getSubvector(data.C, startPos, endPos);
            int[] W_H = Data.calculateStep6(ai, di, C_H, X_H);

            // Об'єднання фінального результату
            Data.insertIntoFullW(data, W_H, startPos, endPos);

            // S6 - сигнал про завершення обчислення 6
            data.semaphore_6.release();
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            System.out.println("T4 finish");
        }
    }

    public void initData() {
        data.d = 1;
        for(int i = 0; i < data.MD[0].length; i++) {
            Arrays.fill(data.MD[i], 1);
        }
    }
}
```

Скріншот виконання програми для N = 16.

```
C:\Users\yarik\.jdks\openjdk-21.0.2\bin\java.exe "-javaagent:D:\IntelliJ IDEA 2023.3.4\lib\idea_rt.jar=57670:D:\Intelli
T1 start
T3 start
T2 start
T4 start
T4 finish
T3 finish
T2 finish
[272, 272, 272, 272, 272, 272, 272, 272, 272, 272, 272, 272, 272, 272, 272, 272]
T1 finish
Time 6,15 ms

Process finished with exit code 0
```

Тестування:

Опис комп'ютера:

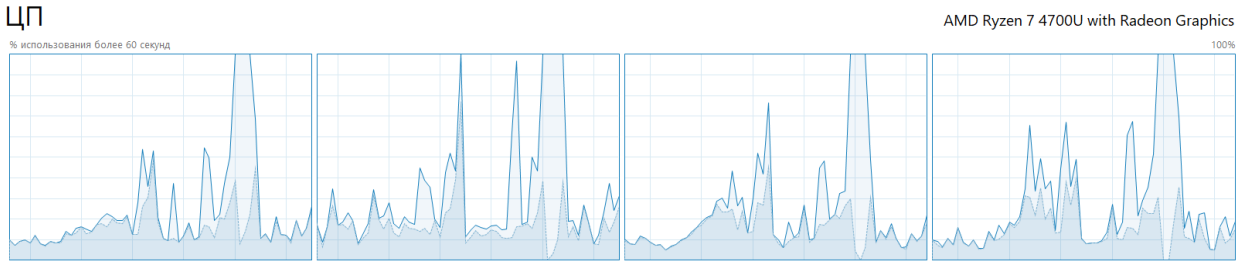
AMD Ryzen 7 4700U with Radeon Graphics

Базовая скорость:	2,00 ГГц
Сокетов:	1
Ядра:	8
Логических процессоров:	8

Для N = 1500.

Час виконання на 1 ядрі: 7842,04 мс ≈ 7,84 с .

Час виконання на 4 ядрах: 3151,05 мс ≈ 3,15 с.



Коефіцієнт прискорення дорівнює:

$$K_y = \frac{T_1}{T_4} = \frac{7{,}84}{3{,}15} = 2{,}49$$

Висновок

1. Лабораторна робота була виконана з використанням мови програмування Java. Для створення багатопотокової програми був використаний клас **Thread**. Для управління семафорами був використаний клас **Semaphore**, для маніпулювання атомік-змінною – **AtomicInteger**, для роботи з бар'єром - **CyclicBarrier**. Крім того, для синхронізації доступу до критичних ділянок був використаний **ReentrantLock** при копіюванні скаляра.
2. Був розроблений паралельний математичний алгоритм для обчислення визначеної формули, що включала множення, додавання матриць, векторів і скалярів, а також визначення максимуму вектора. Було визначено спільні ресурси : a, d, C, MA, E.
3. Було розроблено алгоритми для потоків, де кожен потік виконує паралельне обчислення своєї частини задачі. Також було визначено точки синхронізації та критичні ділянки КД1-КД3.
4. Розроблено структурну схему взаємодії потоків, в якій використані наступні засоби організації:
    a. Семафори використовуються для надсилання та приймання сигналу про завершення обчислення, контролю копіювання спільного ресурсу **a**, а також для синхронізації виведення результату.
    b. Використано атомік-змінну для забезпечення атомарності операції перезапису.
    c. Бар'єр застосовано для синхронізації введення даних.
    d. Критична секція застосована для безпечного копіювання спільного ресурсу **d**.
5. Було проведене тестування, результати якого підтвердили ефективність багатопоточної програми:

При N = 1500. Коефіцієнт прискорення дорівнює: 2,49