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 №4
«Семафори, мютекси, події, критичні секції, бар’єри, атомік-змінні»

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;
- використати як засоби організації взаємодії потоків - монітори
- 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:

R = max(Z)*(B*MV) + e*X*(MM*MC)

1 – MV, MC
2 – MM, R
3 – -
4 – B,X, e, Z

Мова програмування: Java

ОПИС ПРОГРАМИ

І етап. Побудова паралельного математичного алгоритму

Варіант 21

R = max(Z)*(B*MV) + e*X*(MM*MC)

1) ai = max(Zн), i = 1…4

2) a = max(a, ai), СР: a , копія

3) Rн = a*(B*MVн) + e*X*(MM*MCн) СР: a, B, e, MC, X, копія a,e

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

II етап. Розробка алгоритмів потоків

T1:	**Точки синхронізації**
1. Введення MV, MC	
2. Сигнал задачам T2, T3, T4 про введення MV, MC	--S_{2-1} S_{3-1} S_{4-1}
3. Чекати на введення даних в задачах T2, T4	--W_{2-1} W_{4-1}
4. Обчислення 1: a1 = max(Zн)	
5. Обчислення 2: a = max(a, a1)	--КД1
6. Сигнал задачам T2, T3, T4 про завершення обчислення a	--W_{2-2} W_{3-2} W_{4-2}
7. Чекати на завершення обчислення a в задачах T2, T3, T4	--W_{2-2} W_{3-2} W_{4-2}
11. Копія a1 = a	--КД2
12. Копія e1 = e	--КД3
13. Обчислення Rн = a1*(B*MVн) + e1*X*(MM*MCн)	
14. Сигнал задачі T2 про завершення обчислення R	--S_{2-3}
T2:	
1. Введення MM	
2. Сигнал задачам T1, T3, T4 про введення MM	--S_{1-1} S_{3-1} S_{4-1}
3. Чекати на введення даних в задачах T1, T4	-- W_{1-1} W_{4-1}
4. Обчислення1: a2 = max(Zн)	
5. Обчислення2: a = max(a, a2)	--КД1
6. Сигнал задачам T1, T3, T4 про завершення обчислення a	--S_{1-2} S_{3-2} S_{4-2}
7. Чекати на завершення обчислення a в задачах T1, T3, T4	-- W_{1-2} W_{4-2} W_{3-2}
11. Копія a2 = a	-- КД2
12. Копія e2 = e	--КД3
13. Обчислення Rн = a2*(B*MVн) + e2*X*(MM*MCн)	
14. Чекати на завершення обчислення R в задачах T1, T3, T4	-- W_{1-3} W_{4-3} W_{3-3}
15. Виведення R	
T3:	
1. Чекати на введення даних в задачах T1, T2, T4	-- W_{1-1} W_{4-1} W_{2-1}
2. Обчислення1: a3 = max(Zн)	
3. Обчислення2: a = max(a, a3)	КД1

4. Сигнал задачам T1, T2, T4 про завершення обчислення a --$S_{2\text{-}1}$ $S_{1\text{-}1}$ $S_{4\text{-}1}$

5. Чекати на завершення обчислення a в задачах T1, T2, T4 --$W_{2\text{-}2}$ $W_{1\text{-}2}$ $W_{4\text{-}2}$

9. Копія a3 = a --КД2

10. Копія e3 = e --КД3

11. Обчислення Rн = a3*(B*MVн) + e3*X*(MM*MCн)

12. Сигнал задачі T2 про завершення обчислення R --$S_{2\text{-}2}$

T4:

1. Введення B, X, e, Z

2. Сигнал задачам T1, T2, T3 про введення B, X, e, Z --$S_{1\text{-}1}$ $S_{3\text{-}1}$ $S_{2\text{-}1}$

3. Чекати на введення даних в задачах T1, T2 -- $W_{1\text{-}1}$ $W_{2\text{-}1}$

4. Обчислення1: a4 = max(Zн)

5. Обчислення2: a = max(a, a4) --КД1

6. Сигнал задачам T1, T2, T3 про завершення обчислення a --$S_{1\text{-}2}$ $S_{3\text{-}2}$ $S_{2\text{-}2}$

7. Чекати на завершення обчислення a в задачах T1, T2, T3 -- $W_{1\text{-}2}$ $W_{2\text{-}2}$ $W_{3\text{-}2}$

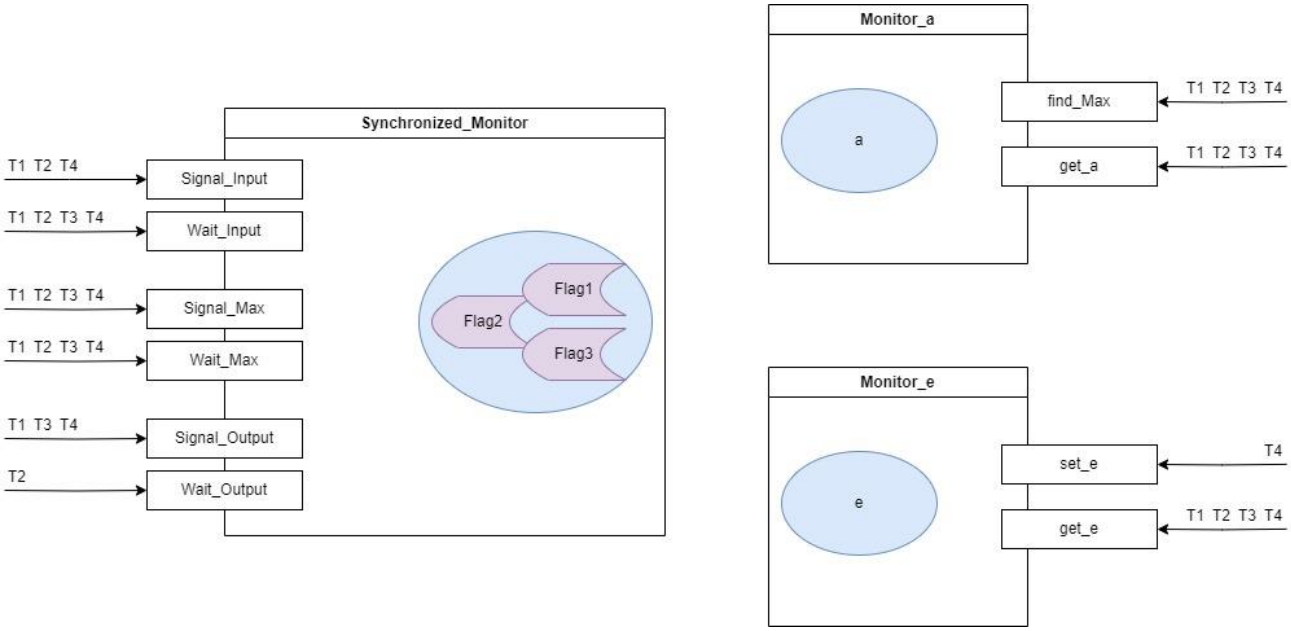
11. Копія a4 = a --КД2

12. Копія e4 = e --КД3

13. Обчислення Rн = a4*(B*MVн) + e4*X*(MM*MCн)

14. Сигнал задачі T2 про завершення обчислення R --$S_{2\text{-}3}$

III Етап. Розробка схеми взаємодії задач



Мал 1 Схема взаємодії задач

Призначення **монітору Synchronized_Monitor -** синхронізація взаємодії потоків

Signal_Input - метод для сигналу про завершення введення даних;
Wait_Input - метод для очікування завершення введення даних;
Signal_Max - метод для сигналу про завершення обчислення max(Z);
Wait_Max - метод для очікування завершення обчислення max(Z);
Signal_Output - метод для сигналу про завершення обчислення $R_н$;
Wait_Output - метод для очікування завершення обчислення $R_н$;

Flag1 – поле прапор синхронізації для введення даних;
Flag2 - поле прапор синхронізації для обчислення max(Z);
Flag3 - поле прапор синхронізації для обчислення $R_н$;

Призначення **монітору Monitor_a** – для вирішення задачі вазємного виключення (захисту спільного ресурсу a)

find_Max – метод для визначення максимального значення змінної a;
get_a - метод для отримання змінної a;

a – поле СР a.

Призначення **монітору Monitor_e** – для вирішення задачі вазємного виключення (захисту спільного ресурсу e)

set_e – метод для задання змінної e;
get_e – метод для отримання змінної e;

e – поле CP e.

IV Етап: Розробка 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 Lab4, допоміжного класу Data, що зберігає всі спільні змінні та містить методи для роботи над векторами і матрицями, клас Synchronized_Monitor, що визначає монітор для синхронізації взаємодії потоків, Monitor_a визначає монітор для вирішення проблеми спільного доступу до ресурсу a, Monitor_e – до ресурсу e.
Також класи Thread_1, Thread_2, Thread_3, Thread_4 відповідають потокам T1, T2, T3, T4 в яких реалізований відповідний алгоритм обчислення

Лістинг програми

Lab4.java

```
package org.example;

//Лабороторна робота ЛР4 Варіант 21
//R = max(Z)*(B*MV) + e*X*(MM*MC)
//T1 - MV, MC
//T2 - MM, R
//T3 -  -
//T4 -  B,X, e, Z
//Дубчак Аліна Євгеніївна ІМ-13
//Дата 28 04 2024


public class Lab4 {
    public static void main(String[] args) {
        Data D = new Data();

        Thread_1 T1 = new Thread_1(1, D);
        Thread_2 T2 = new Thread_2(2, D);
        Thread_3 T3 = new Thread_3(3, D);
        Thread_4 T4 = new Thread_4(4, D);

        T1.start();
        T2.start();
        T3.start();
        T4.start();

        try {
            T1.join();
            T2.join();
            T3.join();
            T4.join();
        } catch (InterruptedException e) {
            throw new RuntimeException(e);
        }

    }
}
```

Data.java

```
package org.example;

public class Data {
    public int N = 8;
    public int P = 4;
    public int H = N / P;
    public int e;
    public int[] Z = new int[N];
    public int[] X = new int[N];
    public int[] B = new int[N];
    public int[] R = new int[N];
    public int[][] MV = new int[N][N];
    public int[][] MC = new int[N][N];
    public int[][] MM = new int[N][N];

    public Synchronized_Monitor Synchronized_Monitor = new
Synchronized_Monitor();
    public Monitor_a Monitor_a = new Monitor_a();
    public Monitor_e Monitor_e = new Monitor_e();

    // Отримання максимального значення вектора
    public static int getMaxValueVector(int[] vector) {
        int max = vector[0];
        for (int i = 0; i < vector.length; i++) {
            if (vector[i] > max) {
                max = vector[i];
            }
        }
        return max;
    }

    // Множення двох матриць
    public static int[][] multiplyTwoMatrices(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;
    }


    // Множення вектора на число
    public static int[] multiplyVectorByNumber(int[] vector, int number) {
        int[] result = new int[vector.length];

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

```
        return result;
    }

    // Множення матриці на вектор
    public static int[] multiplyMatrixByVector(int[][] matrix, int[] vector)
{
        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;

    }

    // Отримання підматриці
    public static int[][] getSubmatrix(int[][] matrix, int start, int end) {
        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;
    }

    // Додавання двох векторів
    public static int[] addTwoVectors(int[] vector1, int[] vector2) {
        int length = vector1.length;
        int[] result = new int[length];

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

        return result;
    }

    // Вставлення підвектора вектору
    public static void insertSubvectorIntoVector(Data data, int[] subv, int
startPos, int endPos) {
        for (int i = startPos, j = 0; i < endPos; i++, j++) {
            data.R[i] = subv[j];
        }
    }

    // Отримання підвектора
    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;
    }

    // Виведення вектора
    public static void printVector(int[] vector) {
        for (int i = 0; i < vector.length; i++) {
            System.out.print(vector[i] + " ");
        }
        System.out.println();
    }

    // Обчислення Rн = a*(B*MVн) + e*Xн*(MMн*MC)
    public static int[] calculateResult(int a, int[] B, int[][] MV, int e,
int[] X, int[][] MM, int[][] MC, int start, int end) {
        int[] res1 = multiplyMatrixByVector(MV, B);
        int[][] res2 = multiplyTwoMatrices(MM, MC, start, end);
        int [] res3 =  multiplyMatrixByVector(res2, X);
        int [] a_res1_h = multiplyVectorByNumber(res1, a);
        int[] e_res3_h = multiplyVectorByNumber(res3, e);
        return addTwoVectors(a_res1_h, e_res3_h);
    }

}
```

Synchronized_Monitor.java

```
package org.example;

public class Synchronized_Monitor {
    private int Flag1 = 0;
    private int Flag2 = 0;
    private int Flag3 = 0;

    public synchronized void Wait_Input() {
        try {
            if (Flag1 < 3) {
                wait();
            }
        } catch (InterruptedException e) {
            throw new RuntimeException(e);
        }
    }

    public synchronized void Signal_Input() {
        Flag1 += 1;
        if (Flag1 == 3) {
            notifyAll();
        }
    }

    public synchronized void Wait_Max() {
        try {
            if (Flag2 < 4) {
                wait();
            }
        } catch (InterruptedException e) {
            throw new RuntimeException(e);
        }

    }
    public synchronized void Signal_Max() {
        Flag2 += 1;
        if (Flag2 == 4) {
```

```
            notifyAll();
        }
    }

    public synchronized void Wait_Output() {
        try {
            if (Flag3 < 2) {
                wait();
            }
        } catch (InterruptedException e) {
            throw new RuntimeException(e);
        }
    }

    public synchronized void Signal_Output() {
        Flag3 += 1;
        if (Flag3 == 3) {
            notify();
        }
    }
}
```

Monitor_a.java

```
package org.example;

public class Monitor_a {
    private int a = Integer.MIN_VALUE;

    public synchronized void find_Max(int b) {
        this.a = Math.max(a, b);
    }

    public synchronized int get_a() {
        return this.a;
    }

}
```

Monitor_e.java

```
package org.example;

public class Monitor_e {
    private int e;

    public synchronized void set_e(int e) {
        this.e = e;
    }

    public synchronized int get_e () {
        return this.e;
    }

}
```

Thread_1.java

```
package org.example;


public class Thread_1 extends Thread {
    private Data data;
```

```
    private int threadId;
    private int start;
    private int end;
    private int a1;
    private int e1;

    public Thread_1(int id, Data D) {
        data = D;
        threadId = id;
        start = (threadId - 1) * data.H;
        end = threadId * data.H;
    }

    // Метод для заповнення початкових даних
    private void fillData() {
        for (int i = 0; i < data.N; i++) {
            for (int j = 0; j < data.N; j++) {
                data.MV[i][j] = 1;
                data.MC[i][j] = 1;
            }
        }
    }

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

            //Сигнал задачі T2, T3, T4 про введення даних
            data.Synchronized_Monitor.Signal_Input();

            // Чекати на введення даних у потоках T2, T4
            data.Synchronized_Monitor.Wait_Input();

            // Обчислення a1 = max(Zн),
            int[] Z_h = Data.getSubvector(data.Z, start, end);
            a1 = Data.getMaxValueVector(Z_h);

            // Обчислення a = max(a, a1)
            data.Monitor_a.find_Max(a1);         //КД1

            // Сигнал T2, T3, T4 про завершення обчислення a
            data.Synchronized_Monitor.Signal_Max();

            // Чекати на завершення обчислень a у потоках T2, T3, T4
            data.Synchronized_Monitor.Wait_Max();
            // Копія a1 = a
            a1 = data.Monitor_a.get_a();          //КД2

            // Копія e1 = e
            e1 = data.Monitor_e.get_e();          //КД3

            int[][] MV_h = Data.getSubmatrix(data.MV, start, end);
            int[][] MC_h = Data.getSubmatrix(data.MC, start, end);

            // Обчислення Rн = a*(B*MVн) + e*Xн*(MMн*MC)
            int[] R_h = Data.calculateResult(a1, data.B, MV_h, e1, data.X,
data.MM, MC_h, start, end);

            // Запис результату
            Data.insertSubvectorIntoVector(data, R_h, start, end);

            // Сигнал про завершення обчислень Rн T2
```

```
            data.Synchronized_Monitor.Signal_Output();

        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            System.out.println("T1 finished");
        }
    }
}
```

Thread_2.java

```
package org.example;

import java.util.Arrays;

public class Thread_2 extends Thread {
    private Data data;
    private int threadId;
    private int start;
    private int end;
    private int a2;
    private int e2;

    public Thread_2(int id, Data D) {
        data = D;
        threadId = id;
        start = (threadId - 1) * data.H;
        end = threadId * data.H;
    }

    // Метод для заповнення початкових даних
    private void fillData() {
        for (int i = 0; i < data.N; i++) {
            for (int j = 0; j < data.N; j++) {
                data.MM[i][j] = 1;
            }
        }
    }

    @Override
    public void run() {

        System.out.println("T2 started");
        try {
            fillData();

            //Сигнал задачі T1, T3, T4 про введення даних
            data.Synchronized_Monitor.Signal_Input();

            // Чекати на введення даних у потоках T1, T4
            data.Synchronized_Monitor.Wait_Input();

            // Обчислення a2 = max(Zн),
            int[] Z_h = Data.getSubvector(data.Z, start, end);
            a2 = Data.getMaxValueVector(Z_h);

            // Обчислення a = max(a, a2)
            data.Monitor_a.find_Max(a2);          //КД1

            // Сигнал T1, T3, T4 про завершення обчислення a
            data.Synchronized_Monitor.Signal_Max();
```

```
            // Чекати на завершення обчислень a у потоках T1, T3, T4
            data.Synchronized_Monitor.Wait_Max();

            // Копія a2 = a
            a2 = data.Monitor_a.get_a();          //КД2

            // Копія e2 = e
            e2 = data.Monitor_e.get_e();          //КД3
            int[][] MV_h = Data.getSubmatrix(data.MV, start, end);
            int[][] MC_h = Data.getSubmatrix(data.MC, start, end);

            // Обчислення Rн = a*(B*MVн) + e*Xн*(MMн*MC)
            int[] R_h = Data.calculateResult(a2, data.B, MV_h, e2, data.X,
data.MM, MC_h, start, end);

            // Запис результату
            Data.insertSubvectorIntoVector(data, R_h, start, end);

            // Чекати на завершення обчислень Rн у потоках T1, T3, T4
            data.Synchronized_Monitor.Wait_Output();

            // Вивід вектора R як результату
            Data.printVector(data.R);
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            System.out.println("T2 finished");
        }
    }
}
```

Thread_3.java

```
package org.example;


public class Thread_3 extends Thread {
    private Data data;
    private int threadId;
    private int start;
    private int end;
    private int a3;
    private int e3;

    public Thread_3(int id, Data D) {
        data = D;
        threadId = id;
        start = (threadId - 1) * data.H;
        end = threadId * data.H;
    }

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

            // Чекати на введення даних у потоках T1, T4, T2
            data.Synchronized_Monitor.Wait_Input();

            // Обчислення a3 = max(Zн),
            int[] Z_h = Data.getSubvector(data.Z, start, end);
            a3 = Data.getMaxValueVector(Z_h);

            // Обчислення a = max(a, a3)
```

```
            data.Monitor_a.find_Max(a3);          //КД1

            int[][] MV_h = Data.getSubmatrix(data.MV, start, end);
            int[][] MC_h = Data.getSubmatrix(data.MC, start, end);

            // Сигнал T2, T1, T4 про завершення обчислення a
            data.Synchronized_Monitor.Signal_Max();

            // Чекати на завершення обчислень a у потоках T2, T1, T4
            data.Synchronized_Monitor.Wait_Max();

            // Копія a3 = a
            a3 = data.Monitor_a.get_a();          //КД2

            // Копія e3 = e
            e3 = data.Monitor_e.get_e();          //КД3

            // Обчислення Rн = a*(B*MVн) + e*Xн*(MMн*MC)
            int[] R_h = Data.calculateResult(a3, data.B, MV_h, e3, data.X,
data.MM, MC_h, start, end);

            // Запис результату
            Data.insertSubvectorIntoVector(data, R_h, start, end);

            // Сигнал про завершення обчислень Rн T2
            data.Synchronized_Monitor.Signal_Output();
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            System.out.println("T3 finished");
        }
    }
}
```

Thread_4.java

```
package org.example;


public class Thread_4 extends Thread {
    private Data data;
    private int threadId;
    private int start;
    private int end;
    private int a4;
    private int e4;

    public Thread_4(int id, Data D) {
        data = D;
        threadId = id;
        start = (threadId - 1) * data.H;
        end = threadId * data.H;
    }

    private void fillData() {
        data.Monitor_e.set_e(1);
        for (int i = 0; i < data.N; i++) {
            data.B[i] = 1;
            data.X[i] = 1;
            data.Z[i] = 1;
        }
    }

    @Override
```

```
    public void run() {
        System.out.println("T4 started");
        try {
            fillData();

            //Сигнал задачі T1, T3, T2 про введення даних
            data.Synchronized_Monitor.Signal_Input();

            // Чекати на введення даних у потоках T1, T2
            data.Synchronized_Monitor.Wait_Input();

            // Обчислення a4 = max(Zн),
            int[] Z_h = Data.getSubvector(data.Z, start, end);
            a4 = Data.getMaxValueVector(Z_h);

            // Обчислення a = max(a, a4)
            data.Monitor_a.find_Max(a4);          //КД1

            // Сигнал T2, T3, T1 про завершення обчислення a
            data.Synchronized_Monitor.Signal_Max();

            // Чекати на завершення обчислень a у потоках T2, T3, T1
            data.Synchronized_Monitor.Wait_Max();

            // Копія a4 = a
            a4 = data.Monitor_a.get_a();          //КД2

            // Копія e4 = e
            e4 = data.Monitor_e.get_e();          //КД3

            int[][] MV_h = Data.getSubmatrix(data.MV, start, end);
            int[][] MC_h = Data.getSubmatrix(data.MC, start, end);

            // Обчислення Rн = a*(B*MVн) + e*Xн*(MMн*MC)
            int[] R_h = Data.calculateResult(a4, data.B, MV_h, e4, data.X,
data.MM, MC_h, start, end);

            // Запис результату
            Data.insertSubvectorIntoVector(data, R_h, start, end);

            // Сигнал про завершення обчислень Rн T2
            data.Synchronized_Monitor.Signal_Output();
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            System.out.println("T4 finished");
        }
    }
}
```

Результат виконання програми

```
T1 started
T2 started
T3 started
T4 started
T1 finished
T4 finished
T3 finished
72 72 72 72 72 72 72 72
T2 finished
```

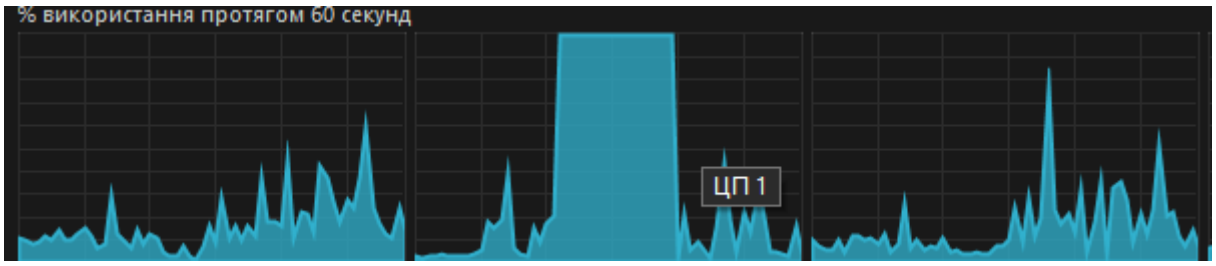
Результат виконання програми при N=8

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 ~ **857ms**
- При навантажені на одне ядро ~ **1951ms**

Кп = **1951ms/ 857ms ~ 2,27**

ВИСНОВОК

1. У процесі розробки було розраховано паралельний математичний алгоритм, який дає змогу виконати обчислення заданої формули та визначити спільні ресурси: a, e, MC, X, B,
2. Був розроблений алгоритм потоків, в яких визначені відповідні точки синхронізації – введення даних, виконання обчислень, виведення. Визначені критичні ділянки(КД1-3).
3. Було створено схему взаємодії задач, де визначено та позначено засоби організації взаємодії потоків: монітори. Розроблено структури класів моніторів Synchronized_Monitor, Monitor_a та Monitor_e.
4. Розробка програми виконувалася на мові Java. Для роботи з потоками було використано клас Threads. У процесі розробки створено монітори: Synchronized_Monitor, Monitor_a та Monitor_e, де для створення синхронізованих методів використовувався synchronized, а також wait() та notify() для блокування/розблокування потоків, модифікатор private для СР.
5. Проведено тестування програми з метою визначення завантаження ядер процесора при N=1000. Значення коефіцієнта прискорення дорівнює 2,27