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

«Програмування потоків»

з дисципліни

«Програмне забезпечення високопродуктивних комп’ютерних систем»

Виконав
студент групи ІМ-13
Кравчук Ілля Володимирович

Перевірив:
доц. Корочкін О. В.

Київ 2024

Завдання

Розробити паралельний алгоритм для рішення математичної задачі, яка буде виконуватися на комп'ютерній системі з чотирма процесорами та двома пристроями вводу-виводу. Описати алгоритм кожного потоку виконання програми з визначенням критичних ділянок і точок синхронізації. Розробити структурну схему взаємодії задач, де використовуються всі вказані засоби взаємодії процесів. Розробити паралельну програму на мові Java, використовуючи семафори для організації взаємодії потоків. Провести налагодження програми та перевірити правильність результатів обчислень.

Варіант 6

A= (R*MC)*MD *p + (B*Z)*E*d

Введення – виведення даних

1: MC, E.

2: MD, d.

3: A, B, p.

4: R, Z.

Етап 1. Побудова паралельного математичного алгоритму.

1) Xн = R * MCн; CP: R;
2) ai = (Bн*Zн);
3) a = a + ai; CP: a;
4) Fн = X * MDн; CP: X;
5) Aн= Fн * p + a * Eн * d; CP: p, a, d – копії;

Захисту потребує CP скаляр "a" при перезаписі і копіюванні та CP скаляри "d" і "p" при копіюванні.

Етап 2. Розробка алгоритмів потоків.

T1	Точки синхронізації
1. Введення MC, E.	
2. *Сигнал* задачам T2, T3, T4 про введення MC, E.	-- $S_{2\text{-}1}$, $S_{3\text{-}1}$, $S_{4\text{-}1}$
3. *Чекати* на введення даних в задачах T2, T3, T4	-- $W_{2\text{-}1}$, $W_{3\text{-}1}$, $W_{4\text{-}1}$
4. Обчислення 1: Xн = R * MCн	
5. Обчислення 2: a1 = (Bн*Zн);	

6. Обчислення 3: a = a + a1 -- КД1
7. *Сигнал* задачам T2, T3, T4 про завершення обчислення 3 -- S_{2-2}, S_{3-2}, S_{4-2}
8. *Чекати* завершення обчислення 3 в задачах T2, T3, T4 -- W_{2-2}, W_{3-2}, W_{4-2}
9. Обчислення 4: Fн = X * MDн
10. *Копіювання* p1 = p -- КД2
11. *Копіювання* a1 = a -- КД3
12. *Копіювання* d1 = d -- КД4
13. Обчислення 5: Ан= Fн * p1 + a1 * Ен * d1
14. *Сигнал* задачі T3 про завершення обчислення 5 -- S_{3-3}

T2 Точки синхронізації

1. Введення MD, d
2. *Сигнал* задачам T1, T3, T4 про введення MD, d -- S_{1-1}, S_{3-1}, S_{4-1}
3. *Чекати* на введення даних в задачах T1, T3, T4 -- W_{1-1}, W_{3-1}, W_{4-1}
4. Обчислення 1: Xн = R * MCн
5. Обчислення 2: a2 = (Bн*Zн);
6. Обчислення 3: a = a + a2 -- КД1
7. *Сигнал* задачам T1, T3, T4 про завершення обчислення 3 -- S_{1-2}, S_{3-2}, S_{4-2}
8. *Чекати* завершення обчислення 3 в задачах T1, T3, T4 -- W_{1-2}, W_{3-2}, W_{4-2}
9. Обчислення 4: Fн = X * MDн
10. *Копіювання* p2 = p -- КД2
11. *Копіювання* a2 = a -- КД3
12. *Копіювання* d2 = d -- КД4
13. Обчислення 5: Ан= Fн * p2 + a2 * Ен * d2
14. *Сигнал* задачі T3 про завершення обчислення 5 -- S_{3-3}

T3 Точки синхронізації

1. Введення A, B, p.
2. *Сигнал* задачам T1, T2, T4 про введення A, B, p. -- S_{1-1}, S_{2-1}, S_{4-1}
3. *Чекати* на введення даних в задачах T1, T2, T4 -- W_{1-1}, W_{2-1}, W_{4-1}
4. Обчислення 1: Xн = R * MCн
5. Обчислення 2: a3 = (Bн*Zн);
6. Обчислення 3: a = a + a3 -- КД1
7. *Сигнал* задачам T1, T2, T4 про завершення обчислення 3 -- S_{1-2}, S_{2-2}, S_{4-2}
8. *Чекати* завершення обчислення 3 в задачах T1, T2, T4 -- W_{1-2}, W_{2-2}, W_{4-2}
9. Обчислення 4: Fн = X * MDн
10. *Копіювання* p3 = p -- КД2
11. *Копіювання* a3 = a -- КД3
12. *Копіювання* d3 = d -- КД4
13. Обчислення 5: Ан= Fн * p3 + a3 * Ен * d3
14. *Чекати* завершення обчислення 5 в задачах T1, T2, T4 -- W_{1-3}, W_{2-3}, W_{4-3}
15. Виведення результату A

T4 Точки синхронізації

1. Введення R, Z.
2. *Сигнал* задачам T1, T2, T3 про введення R, Z. -- S_{1-1}, S_{2-1}, S_{3-1}

3. *Чекати* на введення даних в задачах T1, T2, T3 -- W_{1-1}, W_{2-1}, W_{3-1}
4. Обчислення 1: Xн = R * MCн
5. Обчислення 2: a4 = (Bн*Zн);
6. Обчислення 3: a = a + a4 -- КД1
7. *Сигнал* задачам T1, T2, T3 про завершення обчислення 3 -- S_{1-2}, S_{2-2}, S_{3-2}
8. *Чекати* завершення обчислення 3 в задачах T1, T2, T3 -- W_{1-2}, W_{2-2}, W_{3-2}
9. Обчислення 4: Fн = X * MDн
10. *Копіювання* p4 = p -- КД2
11. *Копіювання* a4 = a -- КД3
12. *Копіювання* d4 = d -- КД4
13. Обчислення 5: Aн= Fн * p4 + a4 * Eн * d4
14. *Сигнал* задачі T3 про завершення обчислення 5 -- S_{3-3}

Етап 3. Розробка схеми взаємодії задач.

При створенні структурної схеми взаємодії задач(Рис.1), були враховані різні методи синхронізації, які доступні в мові Java, такі як використання семафорів, критичних секцій, атомік-змінних і бар'єрів. Кожна задача має чотири критичні ділянки (КД), і кожна з них обробляється за допомогою таких засобів: КД1 використовує атомік-змінну, КД2 - семафори, КД3 - критичну секцію, а КД4 - семафори.

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

S6 - семафор для захисту КД2 (копіювання спільного ресурсу).

S7 - семафор для захисту КД4 (копіювання спільного ресурсу).

S5 - семафор для сигналів до T3 про завершення задачами обчислення 5.

B1 - бар’єр для синхронізації введення даних з потоків T1, T2, T3, T4.
a - атомік-змінна для захисту КД1 (перезапис спільного ресурсу).
CS1 - критична секція для захисту КД3 (копіювання спільного ресурсу).

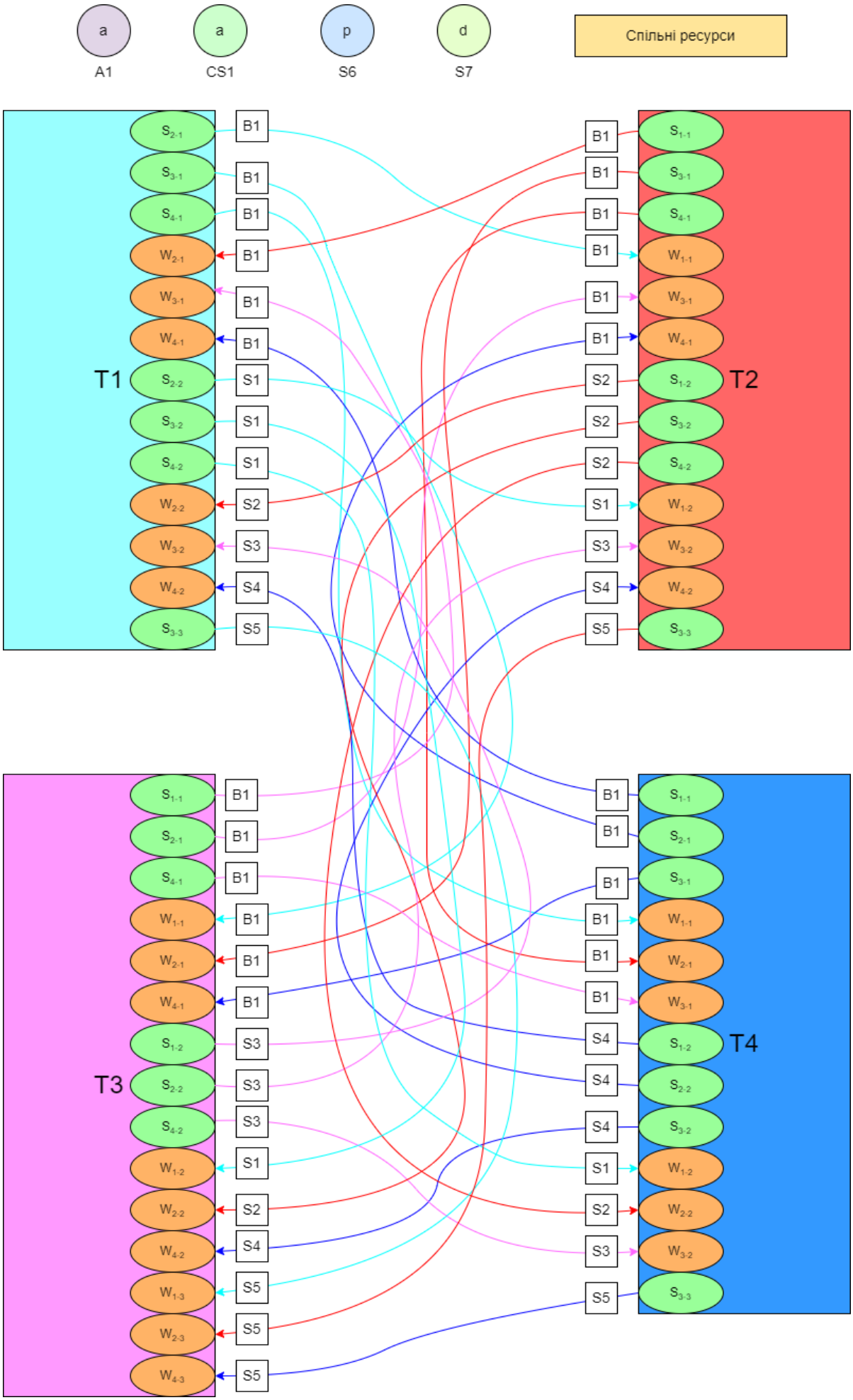

Рис.1 Схеми взаємодії задач

Етап 4. Розроблення програми.

На етапі 2 були розроблені алгоритми, а на етапі 3 була створена структурна схема взаємодії завдань. На їх основі було створено програму на мові Java. Ця програма складається з головного класу Lab2, класу Data для зберігання загальних змінних та статичних методів для роботи з векторами і матрицями, а також класів T1, T2, T3, T4 для відповідних потоків.

Lab2.java

```
// ПЗВПКС
// Лабораторна робота №2
// Варіант 6
// A = (R*MC)*MD *p + (B*Z)*E*d
// Кравчук Ілля Володимирович ІМ-13
// Дата 30.03.2024

public class Lab2 {
    public static void main(String[] args) {
        long startTime = System.nanoTime();
        Data data = new Data();

        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();

        try {
            t1.join();
            t2.join();
            t3.join();
            t4.join();
        } catch (InterruptedException e) {
            System.out.println(e);
        }

        long endTime = System.nanoTime();
        double totalTime = (endTime - startTime) / 1_000_00.0;
        System.out.printf("Час виконання програми: %.1f мілісекунд\n",
totalTime);

    }
}
```

Data.java

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

public class Data {

    public int N = 24;
    public int P = 4;
    public int H = N / P;
    public int d;
    public int p;
```

```
    public int[] A = new int[N];
    public int[] X = new int[N];
    public int[] B = new int[N];
    public int[] E = new int[N];
    public int[] R = new int[N];
    public int[] Z = new int[N];
    public int[][] MC = new int[N][N];
    public int[][] MD = new int[N][N];
    Semaphore S1 = new Semaphore(0);
    Semaphore S2 = new Semaphore(0);
    Semaphore S3 = new Semaphore(0);
    Semaphore S4 = new Semaphore(0);
    Semaphore S5 = new Semaphore(0);
    Semaphore S6 = new Semaphore(1);
    Semaphore S7 = new Semaphore(1);
    CyclicBarrier B1 = new CyclicBarrier(4);
    public AtomicInteger a = new AtomicInteger(0);
    private final Object CS1 = new Object();

    public int initializeVariableToOne(int variable) {
        return 1;
    }

    public int copy_a_CS1() {
        synchronized (CS1) {
            return this.a.intValue();
        }
    }

    public int[][] initializeVectorWithOnes(int[][] matrix) {
        int[][] filledMatrix = new int[matrix.length][matrix[0].length];
        for (int i = 0; i < matrix.length; i++) {
            for (int j = 0; j < matrix[i].length; j++) {
                filledMatrix[i][j] = 1;
            }
        }
        return filledMatrix;
    }
    public int[] initializeVectorWithOnes(int[] vector) {
        int[] filledVector = new int[vector.length];
        for (int i = 0; i < vector.length; i++) {
            filledVector[i] = 1;
        }
        return filledVector;
    }

    public static int[][] subMatrix(int[][] matrix, int startIndex, int
endIndex) {
        int rows = matrix.length;
        if (rows == 0 || startIndex < 0 || endIndex <= startIndex || endIndex >
matrix[0].length) {
            throw new IllegalArgumentException("Invalid submatrix indices");
        }

        int cols = endIndex - startIndex;
        if (cols <= 0) {
            throw new IllegalArgumentException("Invalid submatrix column
range");
        }

        int[][] submatrix = new int[rows][cols];

        for (int i = 0; i < rows; i++) {
            if (matrix[i].length < endIndex) {
                throw new IllegalArgumentException("Invalid row length for
```

```
submatrix");
            }
            System.arraycopy(matrix[i], startIndex, submatrix[i], 0, cols);
        }

        return submatrix;
    }

    public static int[] subVector(int[] vector, int startIndex, int endIndex) {
        if (vector.length == 0 || startIndex < 0 || endIndex <= startIndex ||
endIndex > vector.length) {
            throw new IllegalArgumentException("Invalid subvector indices");
        }

        int size = endIndex - startIndex;
        if (size <= 0) {
            throw new IllegalArgumentException("Invalid subvector range");
        }

        int[] subvector = new int[size];

        System.arraycopy(vector, startIndex, subvector, 0, size);

        return subvector;
    }



    public static int scalarProduct(int[] vector1, int[] vector2) {
        int minLen = Math.min(vector1.length, vector2.length);
        int result = 0;

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

        return result;
    }

    public static int[] vectorMatrixMultiplication(int[] vector, int[][] matrix)
{
        int rows = matrix.length;
        int cols = matrix[0].length;
        int[] result = new int[cols];

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

        return result;
    }
}
```

T1.java

```
import java.util.concurrent.BrokenBarrierException;

class T1 extends Thread {
    private Data data;
    final int id = 1;
    private int a1;
    private int d1;
    private int p1;
```

```
    public T1(Data d) {
        data = d;
    }

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

            data.MC =data.initializeVectorWithOnes(data.MC);
            data.E = data.initializeVectorWithOnes(data.E);


            // Сигнал задачам T2, T3, T4 про введення MC, E і чекати введення
даних в задачах T2, T3, T4 - бар'ер B1
            data.B1.await();

            int indexStart = (id - 1) * data.H;
            int indexEnd =  id * data.H;
            int [][] MCh = Data.subMatrix(data.MC, indexStart, indexEnd);

            // Обчислення 1
            int []Xh = Data.vectorMatrixMultiplication(data.R, MCh);
            for (int i = indexStart; i < indexEnd; i++) {
                data.X[i] += Xh[i - indexStart];
            }
            // Обчислення 2
            int []Bh = Data.subVector(data.B,indexStart, indexEnd);
            int []Zh = Data.subVector(data.Z,indexStart, indexEnd);

            a1 = data.scalarProduct(Bh, Zh);

            // доступ до спільного ресурсу - КД1
            // Обчислення 3
            data.a.updateAndGet(current -> current + a1);          // атомік-
змінна a
            // сигнал про завершення обчислення
            data.S1.release(3);      // семафор S2
            // очікування на завершення обчислень a з T2, T3, T4
            data.S2.acquire();              // семафор S2
            data.S3.acquire();              // семафор S3
            data.S4.acquire();              // семафор S4

            int [][] MDh = Data.subMatrix(data.MD, indexStart, indexEnd);
            // Обчислення 4
            int []Fh = Data.vectorMatrixMultiplication(data.X, MDh);
            int []Eh = Data.subVector(data.E,indexStart, indexEnd);

            // копіювання p1 = p --КД2 семафор S6
            data.S6.acquire();
            p1 = data.p;
            data.S6.release();

            // копіювання a1 = a --КД3 критична секція CS1
            a1 = data.copy_a_CS1();

            // копіювання d1 = d --КД4 семафор S7
            data.S7.acquire();
            d1 = data.d;
            data.S7.release();

            //Обчислення 5 : Ан= Fн * p + a * Eн * d
            int minLen = Math.min(Fh.length, Eh.length);
            int[] Ah = new int[minLen];

            for (int i = 0; i < minLen; i++) {
```

```
                Ah[i] = Fh[i] * p1 + a1 * Eh[i] * d1;
            }

            for (int i = indexStart; i < indexEnd; i++) {
                data.A[i] += Ah[i - indexStart];
            }
            // Сигнал про завершення обчислення A семафор S5
            data.S5.release();


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

T2.java

```
import java.util.concurrent.BrokenBarrierException;

public class T2 extends Thread {
    private Data data;
    final int id = 2;
    private int a2;
    private int d2;
    private int p2;
    public T2(Data d) {
        data = d;
    }

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

        try {
            data.MD = data.initializeVectorWithOnes(data.MD);
            data.d = data.initializeVariableToOne(data.d);

            // Сигнал задачам T1, T3, T4 про введення MD, d і чекати введення
даних в задачах T1, T3, T4 - бар'єр B1
            data.B1.await();

            int indexStart = (id - 1) * data.H;
            int indexEnd =  id * data.H;
            int [][] MCh = Data.subMatrix(data.MC, indexStart, indexEnd);

            // Обчислення 1
            int []Xh = Data.vectorMatrixMultiplication(data.R, MCh);
            for (int i = indexStart; i < indexEnd; i++) {
                data.X[i] += Xh[i - indexStart];
            }
            // Обчислення 2
            int []Bh = Data.subVector(data.B,indexStart, indexEnd);
            int []Zh = Data.subVector(data.Z,indexStart, indexEnd);

            a2 = data.scalarProduct(Bh, Zh);

            // доступ до спільного ресурсу - КД1
            // Обчислення 3
            data.a.updateAndGet(current -> current + a2);          // атомік-
змінна a
            // сигнал про завершення обчислення
            data.S2.release(3);      // семафор S1
            // очікування на завершення обчислень a з T1, T3, T4
```

```
            data.S1.acquire();              // семафор S1
            data.S3.acquire();              // семафор S3
            data.S4.acquire();              // семафор S4

            int [][] MDh = Data.subMatrix(data.MD, indexStart, indexEnd);
            // Обчислення 4
            int []Fh = Data.vectorMatrixMultiplication(data.X, MDh);
            int []Eh = Data.subVector(data.E,indexStart, indexEnd);

            // копіювання p2 = p --КД2 семафор S6
            data.S6.acquire();
            p2 = data.p;
            data.S6.release();

            // копіювання a2 = a --КД3 критична секція CS1
            a2 = data.copy_a_CS1();

            // копіювання d2 = d --КД4 семафор S7
            data.S7.acquire();
            d2 = data.d;
            data.S7.release();

            //Обчислення 5 : Ан= Fн * p + a * Eн * d
            int minLen = Math.min(Fh.length, Eh.length);
            int[] Ah = new int[minLen];

            for (int i = 0; i < minLen; i++) {
                Ah[i] = Fh[i] * p2 + a2 * Eh[i] * d2;
            }

            for (int i = indexStart; i < indexEnd; i++) {
                data.A[i] += Ah[i - indexStart];
            }
            // Сигнал про завершення обчислення A семафор S5
            data.S5.release();

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

T3.java

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

public class T3 extends Thread {
    private Data data;
    final int id = 3;

    private int a3;

    private int d3;
    private int p3;
    public T3(Data d) {
        data = d;
    }

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

        try {
```

```
            data.B = data.initializeVectorWithOnes(data.B);
            data.p = data.initializeVariableToOne(data.p);

            // Сигнал задачам T1, T2, T4 про введення B, p і чекати введення
даних в задачах T1, T2, T4 - бар'єр B1
            data.B1.await();

            int indexStart = (id - 1) * data.H;
            int indexEnd =  id * data.H;
            int [][] MCh = Data.subMatrix(data.MC, indexStart, indexEnd);

            // Обчислення 1
            int []Xh = Data.vectorMatrixMultiplication(data.R, MCh);
            for (int i = indexStart; i < indexEnd; i++) {
                data.X[i] += Xh[i - indexStart];
            }
            // Обчислення 2
            int []Bh = Data.subVector(data.B,indexStart, indexEnd);
            int []Zh = Data.subVector(data.Z,indexStart, indexEnd);

            a3 = data.scalarProduct(Bh, Zh);

            // доступ до спільного ресурсу - КД1
            // Обчислення 3
            data.a.updateAndGet(current -> current + a3);          // атомік-
змінна a
            // сигнал про завершення обчислення
            data.S3.release(3);      // семафор S3
            // очікування на завершення обчислень a з T1, T2, T4
            data.S1.acquire();              // семафор S1
            data.S2.acquire();              // семафор S2
            data.S4.acquire();              // семафор S4

            int [][] MDh = Data.subMatrix(data.MD, indexStart, indexEnd);
            // Обчислення 4
            int []Fh = Data.vectorMatrixMultiplication(data.X, MDh);
            int []Eh = Data.subVector(data.E,indexStart, indexEnd);

            // копіювання p3 = p --КД2 семафор S6
            data.S6.acquire();
            p3 = data.p;
            data.S6.release();

            // копіювання a3 = a --КД3 критична секція CS1
            a3 = data.copy_a_CS1();

            // копіювання d3 = d --КД4 семафор S7
            data.S7.acquire();
            d3 = data.d;
            data.S7.release();

            //Обчислення 5 : Ah= Fh * p + a * Eh * d
            int minLen = Math.min(Fh.length, Eh.length);
            int[] Ah = new int[minLen];

            for (int i = 0; i < minLen; i++) {
                Ah[i] = Fh[i] * p3 + a3 * Eh[i] * d3;
            }

            for (int i = indexStart; i < indexEnd; i++) {
                data.A[i] += Ah[i - indexStart];
            }
            // очікування на завершення обчислень 5 з T1, T2, T4  семафор S5
            data.S5.acquire(3);
            //Вивід A
            System.out.println(Arrays.toString(data.A));
```

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

T4.java

```
import java.util.concurrent.BrokenBarrierException;

public class T4 extends Thread {
    private Data data;
    final int id = 4;
    private int a4;

    private int d4;
    private int p4;

    public T4(Data d) {
        data = d;
    }

    @Override
    public void run() {
        System.out.println("T4 is started");

        try {
            data.R = data.initializeVectorWithOnes(data.R);
            data.Z = data.initializeVectorWithOnes(data.Z);

            // Сигнал задачам T1, T2, T3 про введення R, Z і чекати введення
даних в задачах T1, T2, T3 - бар'єр B1
            data.B1.await();

            int indexStart = (id - 1) * data.H;
            int indexEnd =  id * data.H;
            int [][] MCh = Data.subMatrix(data.MC, indexStart, indexEnd);

            // Обчислення 1
            int []Xh = Data.vectorMatrixMultiplication(data.R, MCh);
            for (int i = indexStart; i < indexEnd; i++) {
                data.X[i] += Xh[i - indexStart];
            }
            // Обчислення 2
            int []Bh = Data.subVector(data.B,indexStart, indexEnd);
            int []Zh = Data.subVector(data.Z,indexStart, indexEnd);

            a4 = data.scalarProduct(Bh, Zh);

            // доступ до спільного ресурсу - КД1
            // Обчислення 3
            data.a.updateAndGet(current -> current + a4);          // атомік-
змінна a
            // сигнал про завершення обчислення
            data.S4.release(3);      // семафор S4
            // очікування на завершення обчислень a з T1, T2, T3
            data.S1.acquire();             // семафор S1
            data.S2.acquire();             // семафор S2
            data.S3.acquire();             // семафор S3

            int [][] MDh = Data.subMatrix(data.MD, indexStart, indexEnd);
            // Обчислення 4
            int []Fh = Data.vectorMatrixMultiplication(data.X, MDh);
```

```
            int []Eh = Data.subVector(data.E,indexStart, indexEnd);

            // копіювання p4 = p --КД2 семафор S6
            data.S6.acquire();
            p4 = data.p;
            data.S6.release();

            // копіювання a4 = a --КД3 критична секція CS1
            a4 = data.copy_a_CS1();

            // копіювання d4 = d --КД4 семафор S7
            data.S7.acquire();
            d4 = data.d;
            data.S7.release();

            //Обчислення 5 : Ан= Fн * p + a * Eн * d
            int minLen = Math.min(Fh.length, Eh.length);
            int[] Ah = new int[minLen];

            for (int i = 0; i < minLen; i++) {
                Ah[i] = Fh[i] * p4 + a4 * Eh[i] * d4;
            }

            for (int i = indexStart; i < indexEnd; i++) {
                data.A[i] += Ah[i - indexStart];
            }
            // Сигнал про завершення обчислення A семафор S5
            data.S5.release();

        } catch (InterruptedException | BrokenBarrierException e) {
            throw new RuntimeException(e);
        } finally {
            System.out.println("T4 is finished");
        }
    }
}
```

Скріншот виконання програми при N = 24 (Рис.2).

```
T1 is started
T4 is started
T2 is started
T3 is started
T2 is finished
T1 is finished
T4 is finished
[600, 600, 600, 600, 600, 600, 600, 600, 600, 600, 600, 600, 600, 600, 600, 600, 600, 600, 600, 600, 600, 600, 600, 600]
T3 is finished

Process finished with exit code 0
```

Рис.2 Скріншот виконання програми

Тестування програми

В ноутбуці є 14 ядер і 20 логічних процесорів(Рис.3).

Рис.3 Характеристики процесора

N = 2500.

Час на одному ядрі 3,02 с.(Рис.4)

```
T1 is started
T2 is started
T4 is started
T3 is started
T2 is finished
T4 is finished
T1 is finished
[6252500, 6252500, 6252500, 6252500, 625250
T3 is finished
Час виконання програми: 3015,2 мілісекунд

Process finished with exit code 0
```

Рис.4 Час виконання програми на одному ядрі.

Час на чотирьох ядрах 1,47 с. (Рис.5)

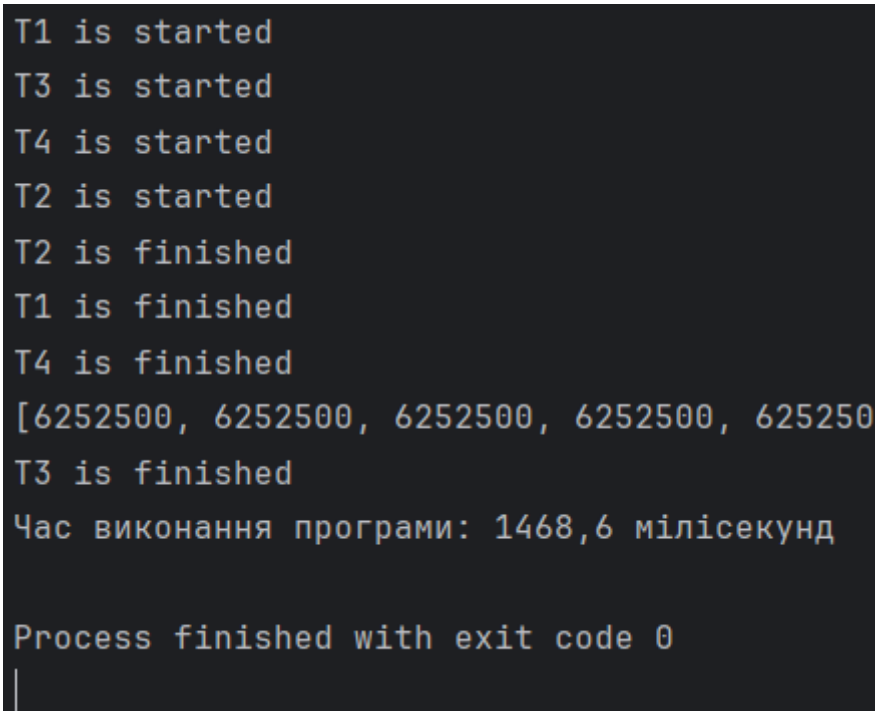

Рис.5 Час виконання на чотирьох ядрах

Кп = 3,02/1,47 = 2,05

Висновок

1.Для створення багатопотокової програми використовувався клас Thread у мові Java, що дозволяє керувати потоками програми.

2.Розроблено паралельний математичний алгоритм, який розділяє математичний вираз на підзадачі для виконання в паралельній системі. У цьому алгоритмі спільними ресурсами є: R, X, p, a, d.

3.Розроблено алгоритм для кожного потоку, у якому визначено точки синхронізації. Ці точки включають синхронізацію введення даних, виконання обчислення і виведення результату. Крім того, визначено чотири завдання взаємного виключення, пов'язані з перезаписом або копіюванням спільних ресурсів.

4.Побудовано структурну схему взаємодії алгоритмів, для якої обрано такі засоби синхронізації:

-Семафори для відправлення та отримання сигналів про завершення обчислення, контролю копіювання спільних ресурсів та синхронізації виведення результату.

-Бар'єр для синхронізації введення даних.

-Атомік-змінну для забезпечення атомарності операцій перезапису.

-Критичну секцію для безпечного копіювання спільного ресурсу.

5. Проведено тестування, під час якого була підтверджена ефективність багатопотокової програми з коефіцієнтом прискорення 2,05.