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

«Монітори»

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 виконання кожного потоку програми, використовуючи засоби організації взаємодії потоків, такі як монітори з використанням synchronized методів. Розробити паралельну програму на мові Java. Провести налагодження програми та перевірити правильність результатів обчислень.

Варіант 18

MX= (B*Z)*(MZ*MM) - (MR*MC)*d

Введення – виведення даних

1: Z, d.

2: MX, MM.

3: MR, B.

4: MC, MZ.

Етап 1. Побудова паралельного математичного алгоритму.

1) ai = (Bн*Zн); i = 1...P
2) a = a + ai; CP: a;
3) MXн = a * (MZ * MMн) - (MR * MCн) * d; CP: a, MZ, MR, d;

N - розмірність вектора/матриці.

P - кількість потоків, які виконують обчислення.

H = N / P

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

Етап 2. Розробка алгоритмів потоків.

T1	Точки синхронізації
1. Ведення Z, d.	
2. *Сигнал* задачам T2, T3, T4 про введення Z, d.	-- $S_{2\text{-}1}$, $S_{3\text{-}1}$, $S_{4\text{-}1}$
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: a1 = (Bн * Zн)	
5. Обчислення 2: a = a + a1	-- КД1
6. *Сигнал* задачам T2, T3, T4 про завершення обчислення 2	-- $S_{2\text{-}2}$, $S_{3\text{-}2}$, $S_{4\text{-}2}$
7. *Чекати* на завершення обчислення 2 в задачах T2, T3, T4	-- $W_{2\text{-}2}$, $W_{3\text{-}2}$, $W_{4\text{-}2}$

8. *Копіювання* a1 = a -- КД2
9. *Копіювання* d1 = d -- КД3
10. Обчислення 3: MXн = a1 * (MZ * MMн) - (MR * MCн) * d1
11. *Сигнал* задачі T2 про завершення обчислення 3 -- S_{2-3}

T2

1. Ведення 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, T3, T4 -- W_{1-1}, W_{3-1}, W_{4-1}
4. Обчислення 1: a2 = (Bн * Zн)
5. Обчислення 2: a = a + a2 -- КД1
6. *Сигнал* задачам T1, T3, T4 про завершення обчислення 2 -- S_{1-2}, S_{3-2}, S_{4-2}
7. *Чекати* на завершення обчислення 2 в задачах T1, T3, T4 -- W_{1-2}, W_{3-2}, W_{4-2}
8. *Копіювання* a2 = a -- КД2
9. *Копіювання* d2 = d -- КД3
10. Обчислення 3: MXн = a2 * (MZ * MMн) - (MR * MCн) * d2
11. *Чекати* на завершення обчислення 3 в задачах T1, T3, T4 -- W_{1-3}, W_{3-3}, W_{4-3}
12. Виведення результату MX

T3

1. Ведення MR, B
2. *Сигнал* задачам T1, T2, T4 про введення MR, B -- 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: a3 = (Bн * Zн)
5. Обчислення 2: a = a + a3 -- КД1
6. *Сигнал* задачам T1, T2, T4 про завершення обчислення 2 -- S_{1-2}, S_{2-2}, S_{4-2}
7. *Чекати* на завершення обчислення 2 в задачах T1, T2, T4 -- W_{1-2}, W_{2-1}, W_{4-2}
8. *Копіювання* a3 = a -- КД2
9. *Копіювання* d3 = d -- КД3
10. Обчислення 3: MXн = a3 * (MZ * MMн) - (MR * MCн) * d3
11. *Сигнал* задачі T2 про завершення обчислення 3 -- S_{2-3}

T4

1. Ведення MR, MZ
2. *Сигнал* задачам T1, T2, T3 про введення MR, MZ -- 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: a4 = (Bн * Zн)
5. Обчислення 2: a = a + a4 -- КД1
6. *Сигнал* задачам T1, T2, T3 про завершення обчислення 2 -- S_{1-2}, S_{2-2}, S_{3-2}

7. *Чекати* на завершення обчислення 2 в задачах T1, T2, T3 -- W_{1-2}, W_{2-1}, W_{3-2}

8. *Копіювання* a4 = a -- КД2

9. *Копіювання* d4 = d -- КД3

10. Обчислення 3: MXн = a4 * (MZ * MMн) - (MR * MCн) * d4

11. *Сигнал* задачі T2 про завершення обчислення 3 -- S_{2-3}

Етап 3. Розробка схеми взаємодії задач.

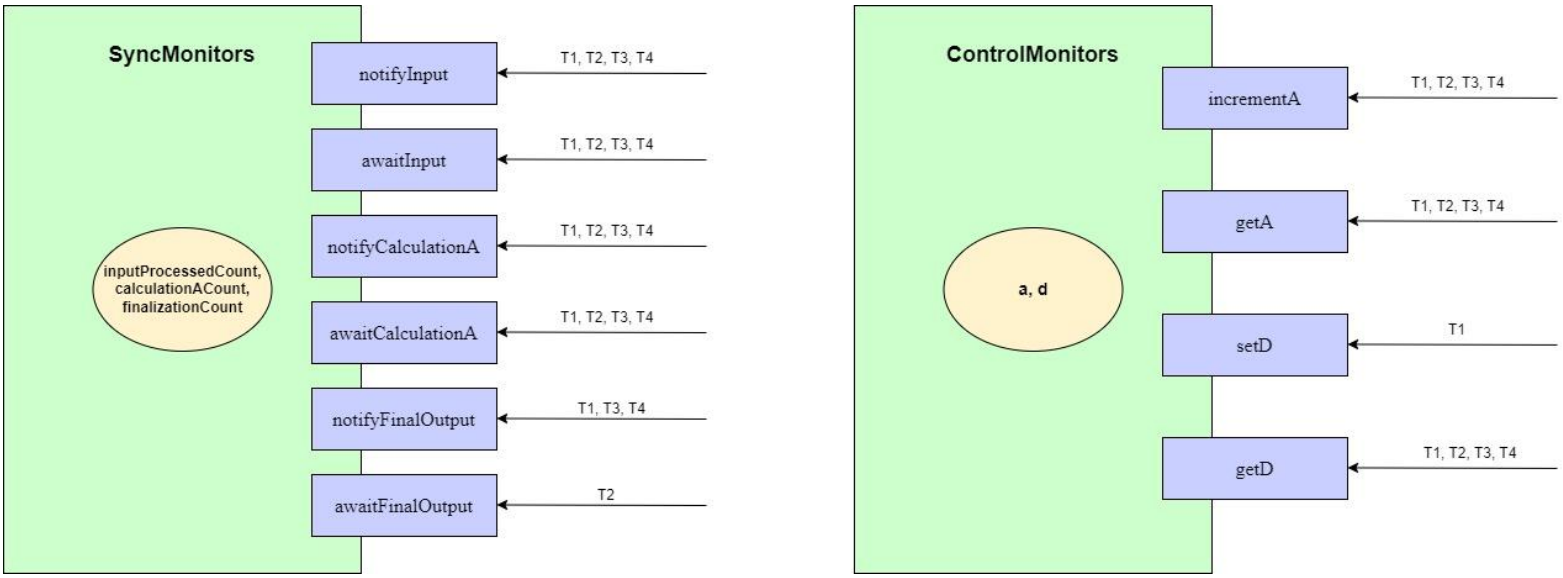

Рис.1 Схеми моніторів

Монітор SyncMonitors забезпечує координацію між потоками(Рис.1).

Методи:

notifyInput: повідомляє про завершення введення даних.

awaitInput: очікує завершення введення даних.

notifyCalculationA: повідомляє про завершення обчислення "a".

awaitCalculationA: очікує завершення обчислення "a".

notifyFinalOutput: повідомляє про завершення обчислення 3.

awaitFinalOutput: очікує завершення обчислення 3.

Змінні:

inputProcessedCount: прапор для синхронізації введення даних.

calculationACount: прапор для синхронізації обчислення "a".

finalizationCount: прапор для синхронізації виведення фінального результату.

Монітор ControlMonitors вирішує проблеми взаємного виключення (Рис.1). Захист спільних ресурсів: "a", "d".

Методи:

setD: задає значення "d".

getD: повертає копію "d".

incrementA: збільшує значення "a" на певне число.

getA: повертає копію "a".

Змінні:

a: CP "a".

d: CP "d".

Етап 4. Розроблення програми.

Головний клас Lab4 запускає чотири потоки (T1, T2, T3, T4), очікує їх завершення та обчислює час виконання. У класі Data оголошені дані та методи для роботи з ними, такі як ініціалізація, обчислення та виведення. Класи ControlMonitors та SyncMonitors забезпечують контроль та синхронізацію потоків. Класи T1, T2, T3, T4 відповідають за роботу потоків, включаючи введення, обчислення та вивід результатів. Код виконує обчислення матриці MX, розподіляючи роботу між потоками та використовуючи монітори для синхронізації.

Lab4.java

```
// ПЗВПКС
// Лабораторна робота №4
// Варіант 18
// MX = (B*Z)*(MZ*MM) - (MR*MC)*d
// Кравчук Ілля Володимирович ІМ-13
// Дата 13.05.2024
public class Lab4 {
    public static void main(String[] args) {
        long startTime = System.nanoTime();

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

        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_000.0; //1мсек=1_000_000нсек
        System.out.printf("Час виконання програми: %.1f мілісекунд\n", totalTime);
    }
}
```

Data.java

```
public class Data {
    public static int N = 15;
    public static int P = 4;
    public static int H = N / P;

    public static int[][] MX = new int[N][N];
    public static int[][] MR = new int[N][N];
    public static int[][] MC = new int[N][N];

    public static int[][] MZ = new int[N][N];
    public static int[][] MM = new int[N][N];

    public static int[] B = new int[N];
    public static int[] Z = new int[N];

    public static ControlMonitors monitorControl = new ControlMonitors();
    public static SyncMonitors monitorSync = new SyncMonitors();

    public static int d;

    public static int initializeVariableToOne() {
        return 1;
    }

    public static int[] initializeVectorWithOnes() {
        int[] filledVector = new int[N];
        for (int i = 0; i < N; i++) {
            filledVector[i] = 1;
        }
        return filledVector;
    }

    public static int[][] initializeMatrixWithOnes() {
        int[][] filledMatrix = new int[N][N];
        for (int i = 0; i < N; i++) {
            for (int j = 0; j < N; j++) {
                filledMatrix[i][j] = 1;
            }
        }
        return filledMatrix;
    }

    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[] 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[][] 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[][] multiplyMatrices(int[][] firstMatrix, int[][] secondMatrix) {
    int rowsInFirst = firstMatrix.length;
    int colsInFirst = firstMatrix[0].length;
    int colsInSecond = secondMatrix[0].length;

    int[][] result = new int[rowsInFirst][colsInSecond];

    for (int i = 0; i < rowsInFirst; i++) {
        for (int j = 0; j < colsInSecond; j++) {
            for (int k = 0; k < colsInFirst; k++) {
                result[i][j] += firstMatrix[i][k] * secondMatrix[k][j];
            }
        }
    }
    return result;
}

public static int[][] multiplyMatrixByNumber(int[][] matrix, int number) {
    int rows = matrix.length;
    int columns = matrix[0].length;

    int[][] result = new int[rows][columns];
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < columns; j++) {
            result[i][j] = matrix[i][j] * number;
        }
    }

    return result;
}

public static int[][] subtractMatrices(int[][] firstMatrix, int[][] secondMatrix) {
    int rows = firstMatrix.length;
    int columns = firstMatrix[0].length;
```

```
        int[][] result = new int[rows][columns];

        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < columns; j++) {
                result[i][j] = firstMatrix[i][j] - secondMatrix[i][j];
            }
        }

        return result;
    }


    public static int[][] calculateStepThree(int[][] MMh, int[][] MCh, int a, int d) {
        int[][] firstResult = multiplyMatrixByNumber(multiplyMatrices(MZ, MMh), a);
        int[][] secondResult = multiplyMatrixByNumber(multiplyMatrices(MR, MCh), d);
        int[][] finalResult = subtractMatrices(firstResult, secondResult);

        return finalResult;
    }

    public static void displayMatrix(int[][] matrix) {
        int rows = matrix.length;
        int columns = matrix[0].length;
        System.out.println("Matrix MX:");
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < columns; j++) {
                System.out.print(matrix[i][j] + "\t");
            }
            System.out.println();
        }
    }

    public static void insertSubMatrix(int[][] subMatrix, int startColumn) {
        int rows = MX.length;
        int subMatrixColumns = subMatrix[0].length;
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < subMatrixColumns; j++) {
                MX[i][startColumn + j] = subMatrix[i][j];
            }
        }
    }

}
```

ControlMonitors.java

```
public class ControlMonitors {
    private int a = 0;
    private int d;

    public synchronized void incrementA(int ai) {
        this.a += ai;
    }

    public synchronized void setD(int d) {
        this.d = d;
    }

    public synchronized int getA() {
        return a;
    }

    public synchronized int getD() {
        return d;
```

```
    }
}
```

SyncMonitors.java

```
public class SyncMonitors {
    private int inputProcessedCount  = 0;
    private int calculationACount   = 0;
    private int finalizationCount  = 0;

    public synchronized void awaitInput() {
        if (inputProcessedCount != 4) {
            try {
                wait();
            } catch (InterruptedException e) {
                throw new RuntimeException(e);
            }
        }
    }

    public synchronized void notifyInput() {
        inputProcessedCount++;
        if (inputProcessedCount == 4) notifyAll();
    }

    public synchronized void awaitCalculationA() {
        if (calculationACount   != 4) {
            try {
                wait();
            } catch (InterruptedException e) {
                throw new RuntimeException(e);
            }
        }
    }

    public synchronized void notifyCalculationA() {
        calculationACount  ++;
        if (calculationACount   == 4) notifyAll();
    }

    public synchronized void awaitFinalOutput() {
        if (finalizationCount  != 3) {
            try {
                wait();
            } catch (InterruptedException e) {
                throw new RuntimeException(e);
            }
        }
    }

    public synchronized void notifyFinalOutput() {
        finalizationCount ++;
        if (finalizationCount  == 3) notify();
    }
}
```

T1.java

```
public class T1 extends Thread {

    @Override
    public void run() {
        System.out.println("T1 started");
        // введення Z, d
```

```
        Data.Z = Data.initializeVectorWithOnes();
        Data.d = Data.initializeVariableToOne();
        Data.monitorControl.setD(Data.d);

        // сигнал задачам T2, T3, T4 про введення Z, d
        Data.monitorSync.notifyInput();
        // очікування введення даних в задачах T2, T3, T4
        Data.monitorSync.awaitInput();

        // Обчислення 1
        int[] Bh = Data.subVector(Data.B, 0, Data.H);
        int[] Zh = Data.subVector(Data.Z, 0, Data.H);

        int a1 = Data.scalarProduct(Bh, Zh);

        // доступ до спільного ресурсу --КД1
        // Обчислення 2
        Data.monitorControl.incrementA(a1);

        // сигнал про завершення обчислення
        Data.monitorSync.notifyCalculationA();
        // очікування на завершення обчислень a в потоках T2, T3, T4
        Data.monitorSync.awaitCalculationA();

        // копіювання a1 = a --КД2
        a1 = Data.monitorControl.getA();

        // копіювання d1 = d --КД3
        int d1 = Data.monitorControl.getD();

        // Обчислення 3 : MXн = a1 *  (MZ * MMн) - (MR * MCн) * d1
        int[][] MMh = Data.subMatrix(Data.MM, 0, Data.H);
        int[][] MCh = Data.subMatrix(Data.MC, 0, Data.H);

        int[][] MXh = Data.calculateStepThree(MMh, MCh, a1, d1);
        // запис результату
        Data.insertSubMatrix(MXh, 0);

        // сигнал про завершення обчислення 3
        Data.monitorSync.notifyFinalOutput();
        System.out.println("T1 finished");
    }
}
```

T2.java

```
public class T2 extends Thread{

    @Override
    public void run() {
        System.out.println("T2 started");
        // введення  MM
        Data.MM = Data.initializeMatrixWithOnes();


        // сигнал задачам T1, T3, T4 про введення  MM
        Data.monitorSync.notifyInput();
        // очікування введення даних в задачах T1, T3, T4
        Data.monitorSync.awaitInput();

        // Обчислення 1
        int[] Bh = Data.subVector(Data.B, Data.H, Data.H * 2);
        int[] Zh = Data.subVector(Data.Z, Data.H, Data.H * 2);

        int a2 = Data.scalarProduct(Bh, Zh);
```

```
        // доступ до спільного ресурсу --КД1
        // Обчислення 2
        Data.monitorControl.incrementA(a2);

        // сигнал про завершення обчислення
        Data.monitorSync.notifyCalculationA();
        // очікування на завершення обчислень а в потоках T1, T3, T4
        Data.monitorSync.awaitCalculationA();

        // копіювання a2 = a --КД2
        a2 = Data.monitorControl.getA();

        // копіювання d2 = d --КД3
        int d2 = Data.monitorControl.getD();

        // Обчислення 3 : MXн = a2 *  (MZ * MMн) - (MR * MCн) * d2
        int[][] MMh = Data.subMatrix(Data.MM, Data.H, Data.H * 2);
        int[][] MCh = Data.subMatrix(Data.MC, Data.H, Data.H * 2);

        int[][] MXh = Data.calculateStepThree(MMh, MCh, a2, d2);
        // запис результату
        Data.insertSubMatrix(MXh, Data.H);

        // очікування на завершення обчислень 3 в T1,T3,T4
        Data.monitorSync.awaitFinalOutput();

        // вивід MX
        Data.displayMatrix(Data.MX);
        System.out.println("T2 finished");
    }
}
```

T3.java

```
public class T3 extends Thread{

    @Override
    public void run() {
        System.out.println("T3 started");
        // введення MR, B
        Data.MR = Data.initializeMatrixWithOnes();
        Data.B = Data.initializeVectorWithOnes();

        // сигнал задачам T1, T2, T4 про введення MR, B
        Data.monitorSync.notifyInput();
        // очікування введення даних в задачах T1, T2, T4
        Data.monitorSync.awaitInput();

        // Обчислення 1
        int[] Bh = Data.subVector(Data.B, Data.H * 2, Data.H * 3);
        int[] Zh = Data.subVector(Data.Z, Data.H * 2, Data.H * 3);

        int a3 = Data.scalarProduct(Bh, Zh);

        // доступ до спільного ресурсу --КД1
        // Обчислення 2
        Data.monitorControl.incrementA(a3);

        // сигнал про завершення обчислення
        Data.monitorSync.notifyCalculationA();
        // очікування на завершення обчислень а в потоках T1, T2, T4
        Data.monitorSync.awaitCalculationA();

        // копіювання a3 = a --КД2
```

```
        a3 = Data.monitorControl.getA();

        // копіювання d3 = d --КД3
        int d3 = Data.monitorControl.getD();

        // Обчислення 3 : МХн = a3 *  (MZ * ММн) - (MR * МСн) * d3
        int[][] MMh = Data.subMatrix(Data.MM, Data.H * 2, Data.H * 3);
        int[][] MCh = Data.subMatrix(Data.MC, Data.H * 2, Data.H * 3);

        int[][] MXh = Data.calculateStepThree(MMh, MCh, a3, d3);
        // запис результату
        Data.insertSubMatrix(MXh, Data.H * 2);

        // сигнал про завершення обчислення 3
        Data.monitorSync.notifyFinalOutput();
        System.out.println("T3 finished");
    }
}
```

T4.java

```
public class T4 extends Thread {

    @Override
    public void run() {
        System.out.println("T4 started");
        // введення MC, MZ
        Data.MC = Data.initializeMatrixWithOnes();
        Data.MZ = Data.initializeMatrixWithOnes();

        // сигнал задачам T1, T2, T3 про введення MC, MZ
        Data.monitorSync.notifyInput();
        // очікування введення даних в задачах T1, T2, T3
        Data.monitorSync.awaitInput();

        // Обчислення 1
        int[] Bh = Data.subVector(Data.B, Data.H * 3, Data.N);
        int[] Zh = Data.subVector(Data.Z, Data.H * 3, Data.N);

        int a4 = Data.scalarProduct(Bh, Zh);

        // доступ до спільного ресурсу --КД1
        // Обчислення 2
        Data.monitorControl.incrementA(a4);

        // сигнал про завершення обчислення
        Data.monitorSync.notifyCalculationA();
        // очікування на завершення обчислень а в потоках T1, T2, T3
        Data.monitorSync.awaitCalculationA();

        // копіювання a4 = a --КД2
        a4 = Data.monitorControl.getA();

        // копіювання d4 = d --КД3
        int d4 = Data.monitorControl.getD();

        // Обчислення 3 : МХн = a4 *  (MZ * ММн) - (MR * МСн) * d4
        int[][] MMh = Data.subMatrix(Data.MM, Data.H * 3, Data.N);
        int[][] MCh = Data.subMatrix(Data.MC, Data.H * 3, Data.N);

        int[][] MXh = Data.calculateStepThree(MMh, MCh, a4, d4);
        // запис результату
        Data.insertSubMatrix(MXh, Data.H * 3);

        // сигнал про завершення обчислення 3
```

```
        Data.monitorSync.notifyFinalOutput();
        System.out.println("T4 finished");
    }
}
```

Скріншот виконання програми при N = 15 (Рис.2).

```
T3 started
T1 started
T2 started
T4 started
T3 finished
T1 finished
T4 finished
Matrix MX:
210 210 210 210 210 210 210 210 210 210 210 210 210 210 210
210 210 210 210 210 210 210 210 210 210 210 210 210 210 210
210 210 210 210 210 210 210 210 210 210 210 210 210 210 210
210 210 210 210 210 210 210 210 210 210 210 210 210 210 210
210 210 210 210 210 210 210 210 210 210 210 210 210 210 210
210 210 210 210 210 210 210 210 210 210 210 210 210 210 210
210 210 210 210 210 210 210 210 210 210 210 210 210 210 210
210 210 210 210 210 210 210 210 210 210 210 210 210 210 210
210 210 210 210 210 210 210 210 210 210 210 210 210 210 210
210 210 210 210 210 210 210 210 210 210 210 210 210 210 210
210 210 210 210 210 210 210 210 210 210 210 210 210 210 210
210 210 210 210 210 210 210 210 210 210 210 210 210 210 210
210 210 210 210 210 210 210 210 210 210 210 210 210 210 210
210 210 210 210 210 210 210 210 210 210 210 210 210 210 210
210 210 210 210 210 210 210 210 210 210 210 210 210 210 210
T2 finished
Час виконання програми: 11,4 мілісекунд

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 = 1500.

Час на одному ядрі 33,3 с.(Рис.4)

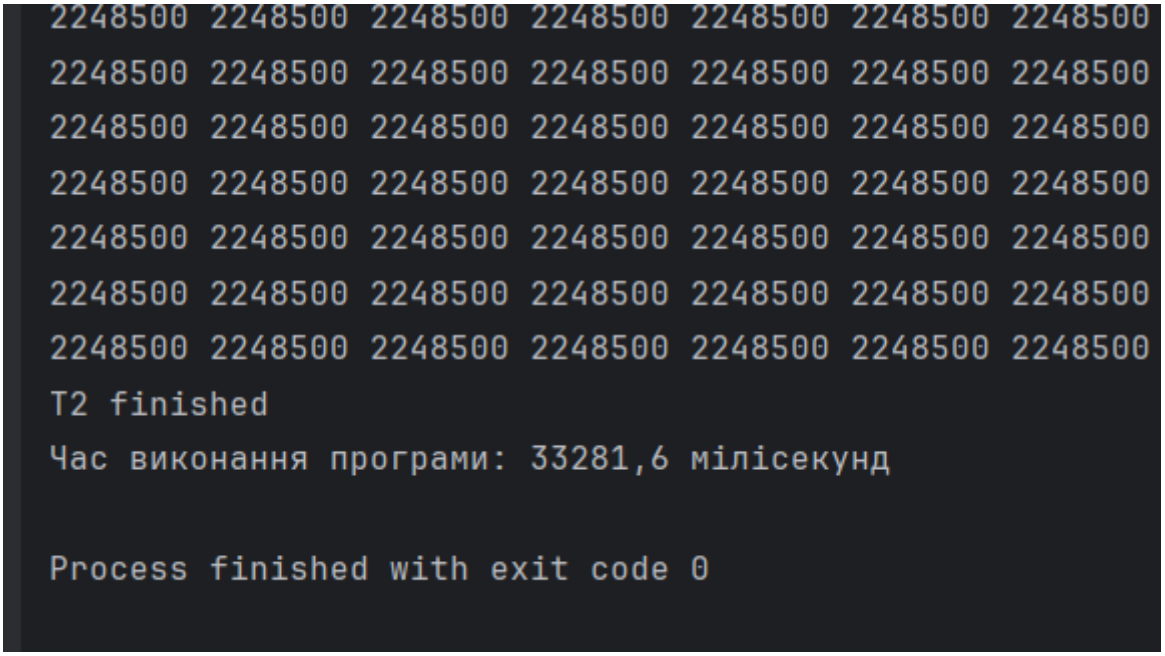

Рис.4 Час виконання програми на одному ядрі.

Час на чотирьох ядрах 15,3 с. (Рис.5)

```
2248500 2248500 2248500 2248500 2248500 2248500 2248500
2248500 2248500 2248500 2248500 2248500 2248500 2248500
2248500 2248500 2248500 2248500 2248500 2248500 2248500
2248500 2248500 2248500 2248500 2248500 2248500 2248500
T2 finished
Час виконання програми: 15305,0 мілісекунд

Process finished with exit code 0
```

Рис.5 Час виконання на чотирьох ядрах

Кп = 33,3/15,3 = 2,19

Висновок

1.Розроблено паралельний математичний алгоритм, який розділяє математичний вираз на підзадачі для виконання в паралельній системі. У цьому алгоритмі спільними ресурсами є: a, MZ, MR, d.

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

3. Створено структурну схему взаємодії потоків, де використані наступні засоби організації: монітор SyncMonitors для синхронізації взаємодії потоків та монітор ControlMonitors для вирішення задачі взаємного виключення.

4. Для створення багатопотокової програми використовувався клас Thread у мові Java, що дозволяє керувати потоками програми. Усього, код реалізує два монітори: SyncMonitors та ControlMonitors, для координації потоків та вирішення проблем взаємного виключення. Монітор SyncMonitors забезпечує синхронізацію потоків щодо введення даних, обчислення "a" та виведення фінального результату, використовуючи прапорці для цієї синхронізації. ControlMonitors, використовується для керування спільними ресурсами "a" і "d", забезпечуючи правильне взаємодію між потоками під час їх доступу до цих ресурсів.

5. Проведено тестування, під час якого була підтверджена ефективність багатопотокової програми з коефіцієнтом прискорення 2,19.