**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»

Виконала:

студентка групи ІМ-11   
Бащак Ярина Володимирівна

Перевірив:

доц. Корочкін О. В.

Київ 2024

**Завдання.**

Побудувати паралельний алгоритм для обчислення формули. Розробити паралельну програму за допомогою мови Java, яка б виконувала це паралельне обчислення за допомогою моніторів. Формула, схема введення та виведення даних потоками обирається згідно варіанту.

Програма має містити класи Lab4, Data та потоки T1, T2, T3, T4.

**Варіант 12:**

Завдання: A= sort(d\*B + Z\*(MM\*MX))\*e + max(B)\*Z

|  |  |  |  |
| --- | --- | --- | --- |
| Введення – виведення даних | | | |
| 1 | 2 | 3 | 4 |
| d, MM | e, B, MX | - | A, Z |

**Виконання роботи.**

**Етап 1. Побудова паралельного математичного алгоритму.**

Розроблено наступний алгоритм:

1. Si = sort(d\*Bн + Z\*(MM\*MXн)), СР: d;  
   де Вн - це H елементів вектора В, МХн – Н стовпців матриці МХ;
2. S = sort(S; Si) СР: S
3. bi = max(Bн)
4. b = max(b, bi) СР: b;
5. Aн = Sн\*e + b\*Zн СР: b, e.

Серед вказаних спільних ресурсів захисту при перезаписі потребують S і b, а при копіюванні – d, b і e.

**Етап 2. Розробка алгоритмів потоків.**

Далі наведено покровий алгоритм для кожної з чотирьох задач.

Задача T1 Точки синхронізації

1. Ведення d, MM
2. Сигнал задачам T2, T3, T4 про введення -- S2-1, S3-1, S4-1
3. Чекати на введення даних в задачах T2, T4 -- W2-1, W4-1
4. Копіювання d1 = d -- КД1
5. Обчислення 1: S1 = sort(d1\*Bн + Z\*(MM\*MXн))
6. Обчислення 2: S = sort(S; S1) -- КД2
7. Обчислення 3: b1 = max(Bн)
8. Обчислення 4: b = max(b, b1) -- КД3
9. Сигнал задачам T2, T3, T4 про завершення обчислення 4 -- S2-2, S3-2, S4-2
10. Чекати на завершення обчислення 4 в задачах T2, T3, T4 -- W2-1, W3-1, W4-2
11. Копіювання b1 = b -- КД4
12. Копіювання e1 = e -- КД5
13. Обчислення 5: Aн = Sн\*e1 + b1\*Zн
14. Сигнал задачі T4 про завершення обчислення 5 -- S4-3

Задача T2 Точки синхронізації

1. Ведення e, B, MX
2. Сигнал задачам T1, T3, T4 про введення -- S1-1, S3-1, S4-1
3. Чекати на введення даних в задачах T1, T4 -- W1-1, W4-1
4. Копіювання d2 = d -- КД1
5. Обчислення 1: S2 = sort(d2\*Bн + Z\*(MM\*MXн))
6. Обчислення 2: S = sort(S; S2) -- КД2
7. Обчислення 3: b2 = max(Bн)
8. Обчислення 4: b = max(b, b2) -- КД3
9. Сигнал задачам T1, T3, T4 про завершення обчислення 4 -- S1-2, S3-2, S4-2
10. Чекати на завершення обчислення 4 в задачах T1, T3, T4 -- W1-2, W3-1, W4-2
11. Копіювання b2 = b -- КД4
12. Копіювання e2 = e -- КД5
13. Обчислення 5: Aн = Sн\*e2 + b2\*Zн
14. Сигнал задачі T4 про завершення обчислення 5 -- S4-3

Задача T3 Точки синхронізації

1. Чекати на введення даних в задачах T1, T2, T4 -- W1-1, W2-1, W4-1
2. Копіювання d3 = d -- КД1
3. Обчислення 1: S3 = sort(d3\*Bн + Z\*(MM\*MXн))
4. Обчислення 2: S = sort(S; S3) -- КД2
5. Обчислення 3: b3 = max(Bн)
6. Обчислення 4: b = max(b, b3) -- КД3
7. Сигнал задачам T1, T2, T4 про завершення обчислення 4 -- S1-1, S2-1, S4-1
8. Чекати на завершення обчислення 4 в задачах T1, T2, T4 -- W1-2, W2-2, W4-2
9. Копіювання b3 = b -- КД4
10. Копіювання e3 = e -- КД5
11. Обчислення 5: Aн = Sн\*e3 + b3\*Zн
12. Сигнал задачі T4 про завершення обчислення 5 -- S4-2

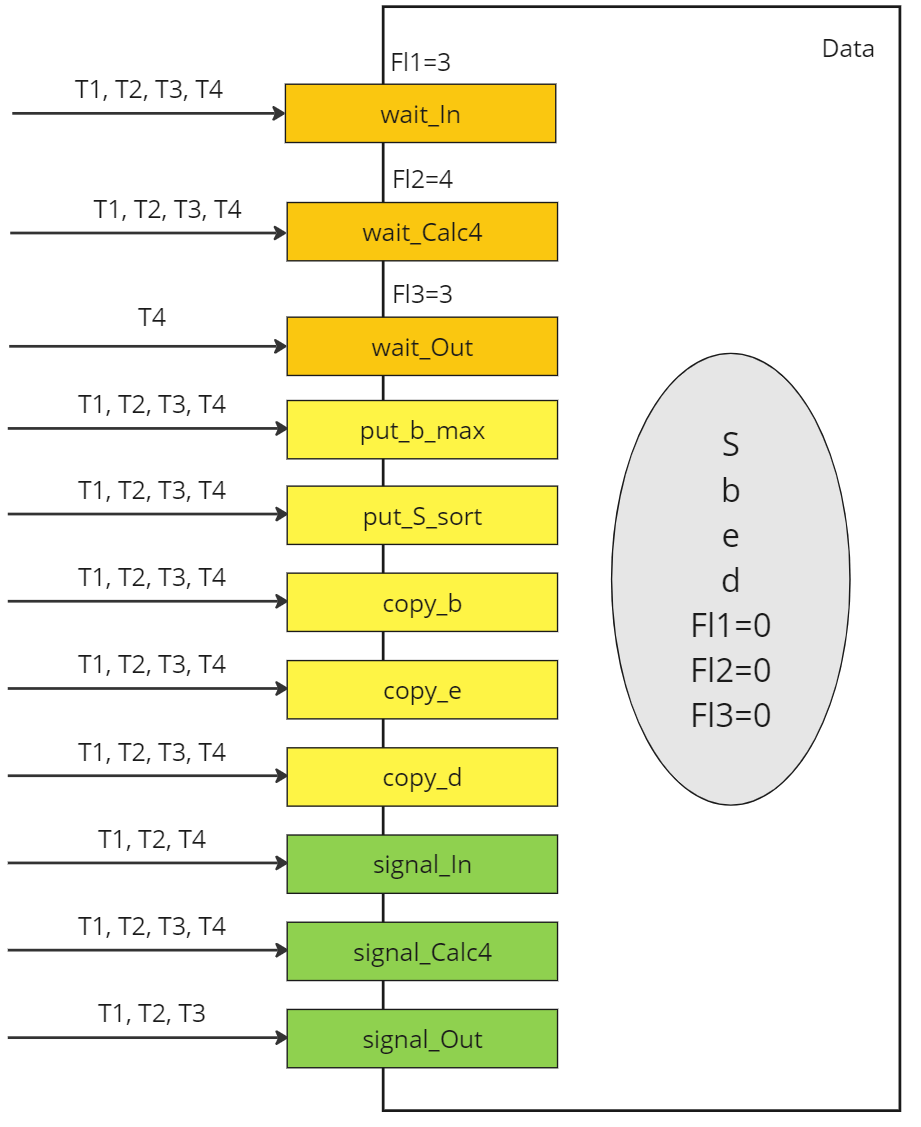
Задача T4 Точки синхронізації

1. Ведення Z
2. Сигнал задачам T2, T3, T1 про введення -- S2-1, S3-1, S1-1
3. Чекати на введення даних в задачах T1, T2 -- W1-1, W2-1
4. Копіювання d4 = d -- КД1
5. Обчислення 1: S4 = sort(d4\*Bн + Z\*(MM\*MXн))
6. Обчислення 2: S = sort(S; S4) -- КД2
7. Обчислення 3: b4 = max(Bн)
8. Обчислення 4: b = max(b, b4) -- КД3
9. Сигнал задачам T2, T3, T1 про завершення обчислення 4 -- S2-2, S3-2, S1-2
10. Чекати на завершення обчислення 4 в задачах T2, T3, T1 -- W2-2, W3-1, W1-2
11. Копіювання b4 = b -- КД4
12. Копіювання e4 = e -- КД5
13. Обчислення 5: Aн = Sн\*e4 + b4\*Zн
14. Чекати на завершення обчислення 5 в задачах T1, T2, T3 -- W1-3, W2-3, W3-2
15. Виведення результату A

**Етап 3. Розробка схеми взаємодії задач**

На мал. 1 зображено схему взаємодії потоків через монітор. В монітор (клас) Data входять спільні ресурси S, b, e, d, а також наступні методи:

* put\_b\_max() – для задання нового значення спільному ресурсу b, якщо дане більше за поточне;
* put\_S\_sort() – для сортування спільного ресурсу S шляхом злиття з новими даними;
* copy\_b(), copy\_e(), copy\_d() – для копіювання спільних ресурсів;
* wait\_In() – для очікування завершення введення даних;
* signal\_In() – для сигналу про завершення введення даних;
* wait\_Calc4() – для очікування завершення обчислення 4;
* signal\_Calc4() – для сигналу про завершення обчислення 4;
* wait\_Out() – для очікування завершення всіх обчислень;
* signal\_Out() – для сигналу про завершення всіх обчислень.



Мал. 1 Схема взаємодії задач через монітор

**Етап 4. Розроблення програми.**

На основі розроблених алгоритмів на етапі 2 і структурної схеми взаємодії задач на етапі 3 було реалізовано програму на мові Java.

Програма складається з основного класу Lab4, класу-монітору Data, а також класів T1, T2, T3, T4 для відповідних потоків. Лістинг програми представлено далі.

// Програмне забезпечення високопродуктивних комп'ютерних систем

// Лабораторна робота №4: монітори

// варіант 12

// A = sort(d\*B + Z\*(MM\*MX))\*e + max(B)\*Z

// Бащак Ярина Володимирівна

// група ІМ-11

// 15.04.2024

public class Lab4 {

    public static void main(String[] args) {

        Data D = new Data();

        Thread1 T1 = new Thread1(1, D);

        Thread2 T2 = new Thread2(2, D);

        Thread3 T3 = new Thread3(3, D);

        Thread4 T4 = new Thread4(4, D);

        T1.start();

        T2.start();

        T3.start();

        T4.start();

    }

}

public class Data {

    public int N = 200;

    public int P = 4;

    public int H = N / P;

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

    public int[][] MX = new int[N][N];

    public int[] A = new int[N];

    public int[] B = new int[N];

    public int[] Z = new int[N];

    private int[] S = new int[N];

    private int b = Integer.MIN\_VALUE;

    private int e;

    private int d;

    private int Fl1 = 0;

    private int Fl2 = 0;

    private int Fl3 = 0;

    public void fill\_data\_T1() {

        d = 1;

        for (int i = 0; i < N; i++) {

            for (int j = 0; j < N; j++) {

                MM[i][j] = 1;

            }

        }

    }

    public void fill\_data\_T2() {

        e = 1;

        for (int i = 0; i < N; i++) {

            B[i] = 1;

            for (int j = 0; j < N; j++) {

                MX[i][j] = 1;

            }

        }

    }

    public void fill\_data\_T4() {

        for (int i = 0; i < N; i++) {

            Z[i] = 1;

        }

    }

    public synchronized int copy\_b() {

        return b;

    }

    public synchronized int copy\_e() {

        return e;

    }

    public synchronized int copy\_d() {

        return d;

    }

    public synchronized void put\_b\_max(int bi) {

        if (bi > b) {

            b = bi;

        }

    }

    public synchronized void put\_S\_sort(int[] new\_Si) {

        int filled = 0;

        for (int value : S) {

            if (value != 0) {

                filled++;

            } else {

                break;

            }

        }

        int[] temp = new int[filled + new\_Si.length];

        int i = 0, j = 0, k = 0;

        while (i < filled && j < new\_Si.length) {

            if (S[i] < new\_Si[j]) {

                temp[k++] = S[i++];

            } else {

                temp[k++] = new\_Si[j++];

            }

        }

        while (i < filled) {

            temp[k++] = S[i++];

        }

        while (j < new\_Si.length) {

            temp[k++] = new\_Si[j++];

        }

        System.arraycopy(temp, 0, S, 0, temp.length);

    }

    public synchronized void wait\_In() {

        try {

            if (Fl1 < 3) {

                wait();

            }

        } catch (Exception e) {

        }

    }

    public synchronized void wait\_Calc4() {

        try {

            if (Fl2 < 4) {

                wait();

            }

        } catch (Exception e) {

        }

    }

    public synchronized void wait\_Out() {

        try {

            if (Fl3 < 3) {

                wait();

            }

        } catch (Exception e) {

        }

    }

    public synchronized void signal\_In() {

        Fl1++;

        if (Fl1 == 3) {

            notifyAll();

        }

    }

    public synchronized void signal\_Calc4() {

        Fl2++;

        if (Fl2 == 4) {

            notifyAll();

        }

    }

    public synchronized void signal\_Out() {

        Fl3++;

        if (Fl3 == 3) {

            notify();

        }

    }

    public static int findMax(int[] vector) {

        int max = vector[0];

        for (int i = 0; i < vector.length; i++) {

            if (vector[i] > max) {

                max = vector[i];

            }

        }

        return max;

    }

    public static int[][] multiplyMatrices(int[][] matrix1, int[][] matrix2) {

        int m = matrix1.length;

        int n = matrix2.length;

        int p = matrix2[0].length;

        int[][] result = new int[m][p];

        for (int i = 0; i < m; i++) {

            for (int j = 0; j < p; j++) {

                result[i][j] = 0;

                for (int k = 0; k < n; k++) {

                    result[i][j] += matrix1[i][k] \* matrix2[k][j];

                }

            }

        }

        return result;

    }

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

        int n = vector.length;

        int m = matrix[0].length;

        int[] result = new int[m];

        for (int i = 0; i < m; i++) {

            result[i] = 0;

            for (int j = 0; j < n; j++) {

                result[i] += vector[j] \* matrix[j][i];

            }

        }

        return result;

    }

    public static int[] multiplyVectorByNumber(int[] vector, int number) {

        int size = vector.length;

        int[] result = new int[size];

        for (int i = 0; i < size; i++) {

            result[i] = vector[i] \* number;

        }

        return result;

    }

    public static int[] addVectors(int[] vector1, int[] vector2) {

        int size = vector1.length;

        int[] result = new int[size];

        for (int i = 0; i < size; i++) {

            result[i] = vector1[i] + vector2[i];

        }

        return result;

    }

    public static int[][] getSubmatrixFromColumns(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;

    }

    public static void insertSubvectorIntoVector(int[] targetVector, int[] subvector, int start) {

        int submatrixSize = subvector.length;

        for (int i = 0; i < submatrixSize; i++) {

            targetVector[start + i] = subvector[i];

        }

    }

    public int[] calculateStep1(int start, int end, int di) {

        int[] B\_H = Arrays.copyOfRange(B, start, end);

        int[][] MX\_H = getSubmatrixFromColumns(MX, start, end);

        int[][] MM\_MX\_H = multiplyMatrices(MM, MX\_H);

        int[] Si = addVectors(multiplyVectorByNumber(B\_H, di), multiplyVectorByMatrix(Z, MM\_MX\_H));

        Arrays.sort(Si);

        return Si;

    }

    public void calculateStep5(int start, int end, int bi, int ei) {

        int[] S\_H = Arrays.copyOfRange(S, start, end);

        int[] Z\_H = Arrays.copyOfRange(Z, start, end);

        int[] A\_H = addVectors(multiplyVectorByNumber(S\_H, ei), multiplyVectorByNumber(Z\_H, bi));

        insertSubvectorIntoVector(A, A\_H, start);

    }

    public static void printVector(int[] vector) {

        for (int i = 0; i < vector.length; i++) {

            System.out.print(vector[i] + " ");

        }

        System.out.println();

    }

}

public class Thread1 extends Thread {

    private Data data;

    private int threadId;

    private int start;

    private int end;

    private int[] S1;

    private int b1;

    private int e1;

    private int d1;

    public Thread1(int id, Data D) {

        data = D;

        threadId = id;

        start = (threadId - 1) \* data.H;

        end = threadId \* data.H;

    }

    @Override

    public void run() {

        System.out.println("T" + threadId + " start");

        try {

            data.fill\_data\_T1();

            // сигнал про введення даних і чекати, щоб інші потоки ввели дані

            data.signal\_In();

            data.wait\_In();

            // копіювання скаляру і обчислення 1

            d1 = data.copy\_d();

            S1 = data.calculateStep1(start, end, d1);

            // обчислення 2

            data.put\_S\_sort(S1);

            // обчислення 3

            int[] B\_H = Arrays.copyOfRange(data.B, start, end);

            b1 = Data.findMax(B\_H);

            // обчислення 4

            data.put\_b\_max(b1);

            // сигнал про завершення обчислення 4 і чекати, щоб інші потоки теж його завершили

            data.signal\_Calc4();

            data.wait\_Calc4();

            // копіювання скалярів і обчислення 5

            b1 = data.copy\_b();

            e1 = data.copy\_e();

            data.calculateStep5(start, end, b1, e1);

            // сигнал про завершення всіх обчислень

            data.signal\_Out();

        } catch (Exception e) {

            e.printStackTrace();

        } finally {

            System.out.println("T" + threadId + " finish");

        }

    }

}

public class Thread2 extends Thread {

    private Data data;

    private int threadId;

    private int start;

    private int end;

    private int[] S2;

    private int b2;

    private int e2;

    private int d2;

    public Thread2(int id, Data D) {

        data = D;

        threadId = id;

        start = (threadId - 1) \* data.H;

        end = threadId \* data.H;

    }

    @Override

    public void run() {

        System.out.println("T" + threadId + " start");

        try {

            data.fill\_data\_T2();

            // сигнал про введення даних і чекати, щоб інші потоки ввели дані

            data.signal\_In();

            data.wait\_In();

            // копіювання скаляру і обчислення 1

            d2 = data.copy\_d();

            S2 = data.calculateStep1(start, end, d2);

            // обчислення 2

            data.put\_S\_sort(S2);

            // обчислення 3

            int[] B\_H = Arrays.copyOfRange(data.B, start, end);

            b2 = Data.findMax(B\_H);

            // обчислення 4

            data.put\_b\_max(b2);

            // сигнал про завершення обчислення 4 і чекати, щоб інші потоки теж його завершили

            data.signal\_Calc4();

            data.wait\_Calc4();

            // копіювання скалярів і обчислення 5

            b2 = data.copy\_b();

            e2 = data.copy\_e();

            data.calculateStep5(start, end, b2, e2);

            // сигнал про завершення всіх обчислень

            data.signal\_Out();

        } catch (Exception e) {

            e.printStackTrace();

        } finally {

            System.out.println("T" + threadId + " finish");

        }

    }

}

public class Thread3 extends Thread {

    private Data data;

    private int threadId;

    private int start;

    private int end;

    private int[] S3;

    private int b3;

    private int e3;

    private int d3;

    public Thread3(int id, Data D) {

        data = D;

        threadId = id;

        start = (threadId - 1) \* data.H;

        end = threadId \* data.H;

    }

    @Override

    public void run() {

        System.out.println("T" + threadId + " start");

        try {

            // чекати, щоб інші потоки ввели дані

            data.wait\_In();

            // копіювання скаляру і обчислення 1

            d3 = data.copy\_d();

            S3 = data.calculateStep1(start, end, d3);

            // обчислення 2

            data.put\_S\_sort(S3);

            // обчислення 3

            int[] B\_H = Arrays.copyOfRange(data.B, start, end);

            b3 = Data.findMax(B\_H);

            // обчислення 4

            data.put\_b\_max(b3);

            // сигнал про завершення обчислення 4 і чекати, щоб інші потоки теж його завершили

            data.signal\_Calc4();

            data.wait\_Calc4();

            // копіювання скалярів і обчислення 5

            b3 = data.copy\_b();

            e3 = data.copy\_e();

            data.calculateStep5(start, end, b3, e3);

            // сигнал про завершення всіх обчислень

            data.signal\_Out();

        } catch (Exception e) {

            e.printStackTrace();

        } finally {

            System.out.println("T" + threadId + " finish");

        }

    }

}

public class Thread4 extends Thread {

    private Data data;

    private int threadId;

    private int start;

    private int end;

    private int[] S4;

    private int b4;

    private int e4;

    private int d4;

    public Thread4(int id, Data D) {

        data = D;

        threadId = id;

        start = (threadId - 1) \* data.H;

        end = threadId \* data.H;

    }

    @Override

    public void run() {

        long startTime = System.currentTimeMillis();

        System.out.println("T" + threadId + " start");

        try {

            data.fill\_data\_T4();

            // сигнал про введення даних і чекати, щоб інші потоки ввели дані

            data.signal\_In();

            data.wait\_In();

            // копіювання скаляру і обчислення 1

            d4 = data.copy\_d();

            S4 = data.calculateStep1(start, end, d4);

            // обчислення 2

            data.put\_S\_sort(S4);

            // обчислення 3

            int[] B\_H = Arrays.copyOfRange(data.B, start, end);

            b4 = Data.findMax(B\_H);

            // обчислення 4

            data.put\_b\_max(b4);

            // сигнал про завершення обчислення 4 і чекати, щоб інші потоки теж його завершили

            data.signal\_Calc4();

            data.wait\_Calc4();

            // копіювання скалярів і обчислення 5

            b4 = data.copy\_b();

            e4 = data.copy\_e();

            data.calculateStep5(start, end, b4, e4);

            // чекати, щоб інші потоки завершили всі обчислення

            data.wait\_Out();

            // вивід результату

            Data.printVector(data.A);

        } catch (Exception e) {

            e.printStackTrace();

        } finally {

            System.out.println("T" + threadId + " finish");

            long endTime = System.currentTimeMillis();

            long totalTime = endTime - startTime;

            System.out.println("Time taken: " + totalTime + " ms");

        }

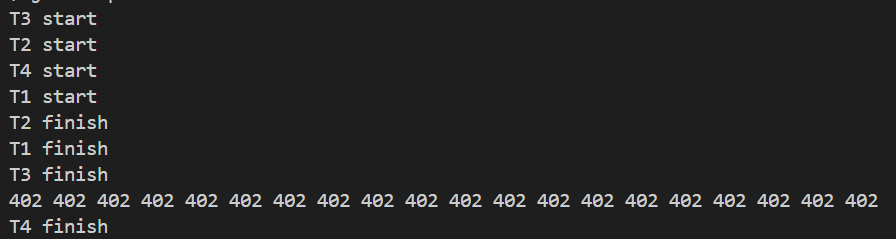
    }

}

**Етап 5. Тестування програми.**

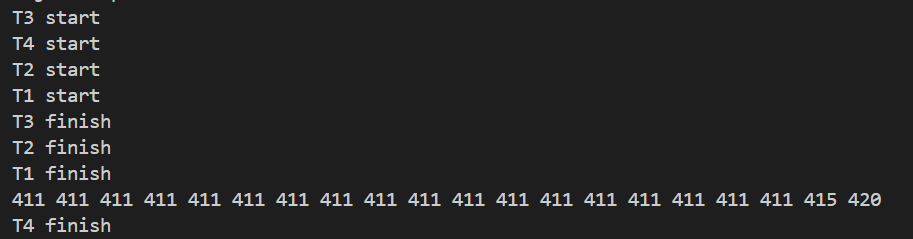
Оскільки формула обчислення міститься такі функції як sort і max, доцільно буде порівняти результати програми, коли всі дані заповнені одиницями з тими, де є не одиничні елементи.

Тому на мал. 2 надано скриншот результату роботи програми для N = 20, коли всі дані заповнені одиницями.



Мал. 2 Вивід програми, коли всі дані заповнені одиницями

А на мал. 3 коли теж всі одиниці, але В[2] = 10 і B[5] = 5. Отримані результати співпадають з теоретичними розрахунками, а отже, програма обчислює формулу правильно.



Мал. 3 Вивід програми, коли В[2] = 10 і B[5] = 5

Також потрібно знайти коефіцієнт прискорення. Тестування програми було проведено на комп’ютері з процесором Intel(R) Core(TM) i5-8265U, ядер 4, логічних процесорів 8.

У рамках експерименту з обчисленням ефективності багатопоточності програми при N=2000, було встановлено, що:

* час виконання програми, коли вона розподіляється системою автоматично по доступним ядрам: 23,3 секунди;
* час при обмеженні виконання лише 1-м ядром: 37,8 секунд.

Отже, Кп = 37,8/23,3 = 1,62.

**Висновки:**

1. розроблено паралельний математичний алгоритм, який дозволяє виконати обчислення заданої формули і визначити, що спільними ресурсами є вектор S і скаляри b, e, d;
2. розроблені алгоритми кожної з задач, визначені завдання синхронізації, до яких входять синхронізація по введенню даних, виконанню обчислення 5 і виведенню результату, а також п’ять завдань взаємного виключення пов'язаних з перезаписом або копіюванням спільних ресурсів;
3. побудовано структурну схему їх взаємодії, де засобами синхронізації є синхронізовані методи класу-монітору;
4. реалізовано багатопоточну програму на мові Java, де для кожної задачі створюється окремий потік;
5. проведено тестування, яке показало ефективність багатопоточної програми: при N = 2000 Кп = 1,62.