**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 №5**

«OpenMP»

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, яка б виконувала це паралельне обчислення на мові Java аналогічно до функціоналу бібліотеки OpenMP. Формула, схема введення та виведення даних потоками обирається згідно варіанту. Програма має містити класи Lab5 і Data.

**Варіант 8:**

Завдання: X= p\*max(C\*(MA\*MD))\*R+ e\*B

|  |  |  |  |
| --- | --- | --- | --- |
| Введення – виведення даних | | | |
| 1 | 2 | 3 | 4 |
| X,e | C, MA | R,MD | B,p |

**Виконання роботи.**

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

Розроблено наступний алгоритм:

1. ai = max(C\*(MA\*MDн)),  
   де МDн – Н стовпців матриці МD;
2. a = max(a, ai) СР: a;
3. Xн = p\*a\*Rн + e\*Bн СР: p, a, e.

Серед вказаних спільних ресурсів захисту при перезаписі потребує a, а при копіюванні – p, a і e.

**Етап 2. Розробка алгоритмів потоків.**

Далі наведено покроковий алгоритм для кожної з чотирьох задач.

Задача T1 Точки синхронізації

1. Ведення e
2. Сигнал задачам T2, T3, T4 про введення -- S2-1, S3-1, S4-1
3. Чекати на введення даних в задачах T2, T3, T4 -- W2-1, W3-1, W4-1
4. Обчислення 1: a1 = max(C\*(MA\*MDн))
5. Обчислення 2: a = max(a, a1) -- КД1
6. Сигнал задачам T2, T3, T4 про завершення обчислення 2 -- S2-2, S3-2, S4-2
7. Чекати на завершення обчислення 2 в задачах T2, T3, T4 -- W2-2, W3-2, W4-2
8. Копіювання p1 = p -- КД2
9. Копіювання a1 = a -- КД3
10. Копіювання e1 = e -- КД4
11. Обчислення 3: Xн = p1\*a1\*Rн + e1\*Bн
12. Чекати на завершення обчислення 3 в задачах T2, T3, T4 -- W2-3, W3-3, W4-3
13. Виведення результату X

Задача T2 Точки синхронізації

1. Ведення C, MA
2. Сигнал задачам T1, T3, T4 про введення -- S1-1, S3-1, S4-1
3. Чекати на введення даних в задачах T1, T3, T4 -- W1-1, W3-1, W4-1
4. Обчислення 1: a2 = max(C\*(MA\*MDн))
5. Обчислення 2: a = max(a, a2) -- КД1
6. Сигнал задачам T1, T3, T4 про завершення обчислення 2 -- S1-2, S3-2, S4-2
7. Чекати на завершення обчислення 2 в задачах T1, T3, T4 -- W1-2, W3-2, W4-2
8. Копіювання p2 = p -- КД2
9. Копіювання a2 = a -- КД3
10. Копіювання e2 = e -- КД4
11. Обчислення 3: Xн = p2\*a2\*Rн + e2\*Bн
12. Сигнал задачі T1 про завершення обчислення 3 -- S1-3

Задача T3 Точки синхронізації

1. Ведення R, MD
2. Сигнал задачам T2, T1, T4 про введення -- S2-1, S1-1, S4-1
3. Чекати на введення даних в задачах T2, T1, T4 -- W2-1, W1-1, W4-1
4. Обчислення 1: a3 = max(C\*(MA\*MDн))
5. Обчислення 2: a = max(a, a3) -- КД1
6. Сигнал задачам T2, T1, T4 про завершення обчислення 2 -- S2-2, S1-2, S4-2
7. Чекати на завершення обчислення 2 в задачах T2, T1, T4 -- W2-2, W1-2, W4-2
8. Копіювання p3 = p -- КД2
9. Копіювання a3 = a -- КД3
10. Копіювання e3 = e -- КД4
11. Обчислення 3: Xн = p3\*a3\*Rн + e3\*Bн
12. Сигнал задачі T1 про завершення обчислення 3 -- S1-3

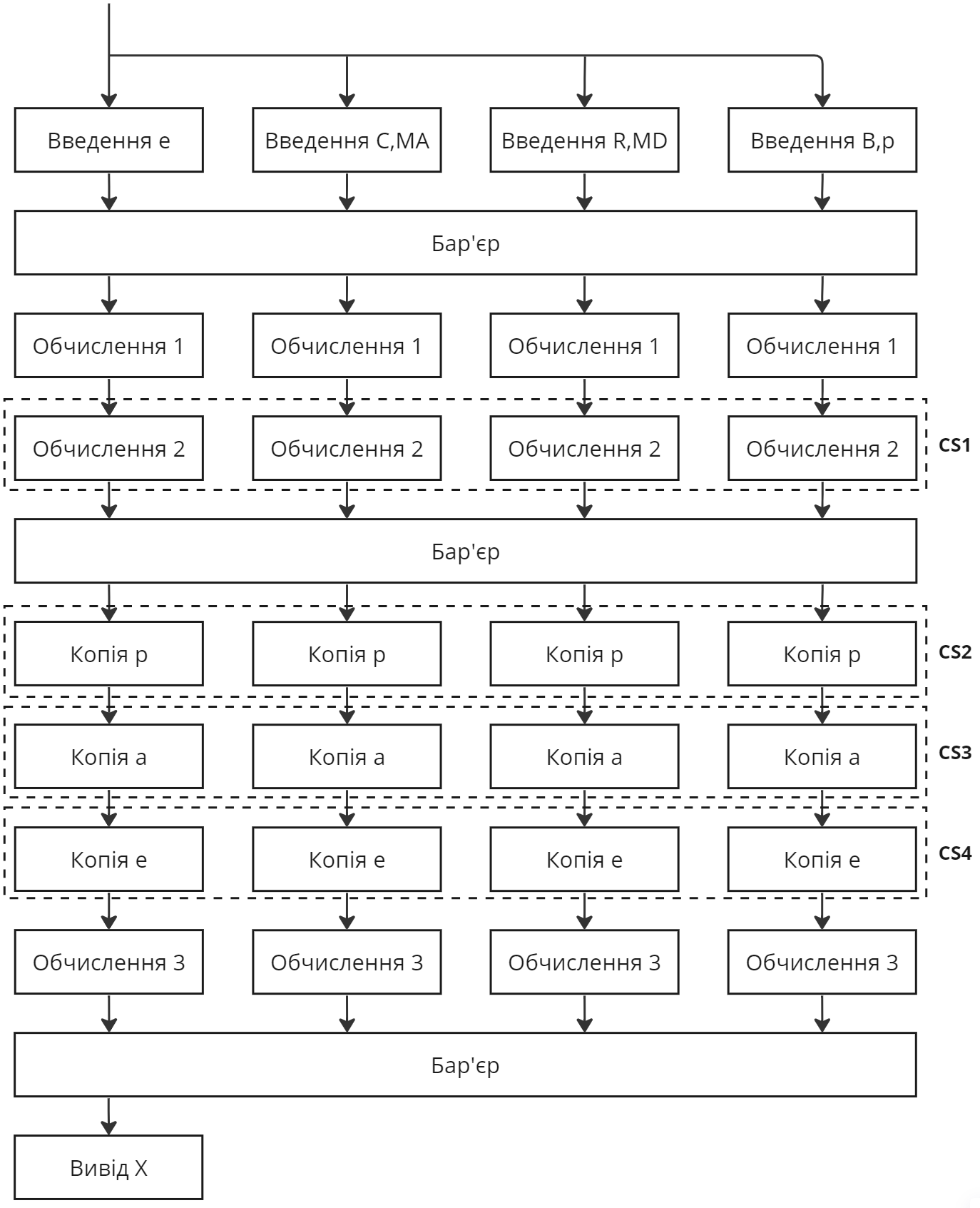
Задача T4 Точки синхронізації

1. Ведення B, p
2. Сигнал задачам T2, T3, T1 про введення -- S2-1, S3-1, S1-1
3. Чекати на введення даних в задачах T2, T3, T1 -- W2-1, W3-1, W1-1
4. Обчислення 1: a4 = max(C\*(MA\*MDн))
5. Обчислення 2: a = max(a, a4) -- КД1
6. Сигнал задачам T2, T3, T1 про завершення обчислення 2 -- S2-2, S3-2, S1-2
7. Чекати на завершення обчислення 2 в задачах T2, T3, T1 -- W2-2, W3-2, W1-2
8. Копіювання p4 = p -- КД2
9. Копіювання a4 = a -- КД3
10. Копіювання e4 = e -- КД4
11. Обчислення 3: Xн = p4\*a4\*Rн + e4\*Bн
12. Сигнал задачі T1 про завершення обчислення 3 -- S1-3

**Етап 3. Розробка схеми взаємодії задач**

На мал. 1 зображено схему взаємодії потоків, виділено одну паралельну ділянку, в якій використано такі засоби синхронізації:

* бар’єри для очікування завершення введення та виведення даних, а також обчислення 2;
* критичні секції для чотирьох критичних ділянок.



Мал. 1 Схема взаємодії задач

**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.

В даній прогамі для імітації паралельної обробки, аналогічної директивам OpenMP, використовується ExecutorService. Цей сервіс дозволяє створювати групу потоків, а за допомогою команди Executors.newFixedThreadPool(P) можна задати їх кількість.

Також в програмі продемонстровано використання аналогу до директиви #pragma omp parallel for. Хоча Java не має прямого еквівалента директивам OpenMP, вона пропонує схожу функціональність. Наступна частина коду демонструє це:

IntStream.range(start, end).parallel().forEach(index -> {

    data.X[index] = pi \* finalAi \* data.R[index] + ei \* data.B[index];

});

IntStream.range(start, end) вказує діапазон індексів, по якому буде ітеруватись цикл, а метод .parallel() активує паралельну обробку потоку. Це дозволяє автоматично розподілити цей діапазон між доступними процесорами, аналогічно до того, як OpenMP розподіляє ітерації циклу між потоками.

Лістинг програми представлено далі.

// Програмне забезпечення високопродуктивних комп'ютерних систем

// Лабораторна робота №5: OpenMP

// варіант 8

// X = p\*max(C\*(MA\*MD))\*R + e\*B

// Бащак Ярина Володимирівна

// група ІМ-11

// 02.05.2024

public class Lab5 {

    public static void main(String[] args) {

        Data data = new Data();

        final long startTime = System.nanoTime();

        for (int i = 0; i < data.P; i++) {

            final int start = i \* data.H;

            int end = (i + 1) \* data.H;

            int threadId = i + 1;

            data.executor.submit(() -> {

                int ai, pi, ei;

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

                // введення даних

                if (threadId == 1) {

                    data.fill\_data\_T1();

                } else if (threadId == 2) {

                    data.fill\_data\_T2();

                } else if (threadId == 3) {

                    data.fill\_data\_T3();

                } else {

                    data.fill\_data\_T4();

                }

                // бар'єр

                try {

                    data.barrier.await();

                } catch (InterruptedException | BrokenBarrierException e) {

                    e.printStackTrace();

                }

                // обчислення 1

                int[][] MD\_H = Data.getSubmatrixFromColumns(data.MD, start, end);

                ai = Data.findMax(Data.multiplyVectorByMatrix(data.C, Data.multiplyMatrices(data.MA, MD\_H)));

                // обчислення 2

                synchronized (data.calc2) {

                    if (ai > data.a) {

                        data.a = ai;

                    }

                }

                // бар'єр

                try {

                    data.barrier.await();

                } catch (InterruptedException | BrokenBarrierException e) {

                    e.printStackTrace();

                }

                // копіювання p, a, e

                synchronized (data.copy\_p) {

                    pi = data.p;

                }

                synchronized (data.copy\_a) {

                    ai = data.a;

                }

                synchronized (data.copy\_e) {

                    ei = data.e;

                }

                // обчислення 3 - імітація поведінки #pragma omp parallel for

                final int finalAi = ai;

                IntStream.range(start, end).parallel().forEach(index -> {

                    data.X[index] = pi \* finalAi \* data.R[index] + ei \* data.B[index];

                });

                // бар'єр

                try {

                    data.barrier.await();

                } catch (InterruptedException | BrokenBarrierException e) {

                    e.printStackTrace();

                }

                // потік 1 виводить вектор Х

                if (threadId == 1) {

                    Data.printVector(data.X);

                }

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

            });

        }

        data.executor.shutdown();

        try {

            if (!data.executor.awaitTermination(1, TimeUnit.HOURS)) {

                System.out.println("Executor did not terminate in the specified time.");

            }

            long endTime = System.nanoTime();

            long totalExecutionTime = (endTime - startTime) / 1\_000\_000;

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

        } catch (InterruptedException e) {

            System.out.println("Threads interrupted: " + e.getMessage());

        }

    }

}

public class Data {

    public int N = 1000;

    public int P = 4;

    public int H = N / P;

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

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

    public int[] B = new int[N];

    public int[] C = new int[N];

    public int[] R = new int[N];

    public int[] X = new int[N];

    public int p;

    public int e;

    public int a = Integer.MIN\_VALUE;

    CyclicBarrier barrier = new CyclicBarrier(P);

    public Object calc2 = new Object();

    public Object copy\_p = new Object();

    public Object copy\_a = new Object();

    public Object copy\_e = new Object();

    ExecutorService executor = Executors.newFixedThreadPool(P);

    public void fill\_data\_T1() {

        e = 1;

    }

    public void fill\_data\_T2() {

        for (int i = 0; i < N; i++) {

            C[i] = 1;

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

                MA[i][j] = 1;

            }

        }

    }

    public void fill\_data\_T3() {

        for (int i = 0; i < N; i++) {

            R[i] = 1;

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

                MD[i][j] = 1;

            }

        }

    }

    public void fill\_data\_T4() {

        p = 1;

        for (int i = 0; i < N; i++) {

            B[i] = 1;

        }

    }

    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[][] 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 printVector(int[] vector) {

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

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

        }

        System.out.println();

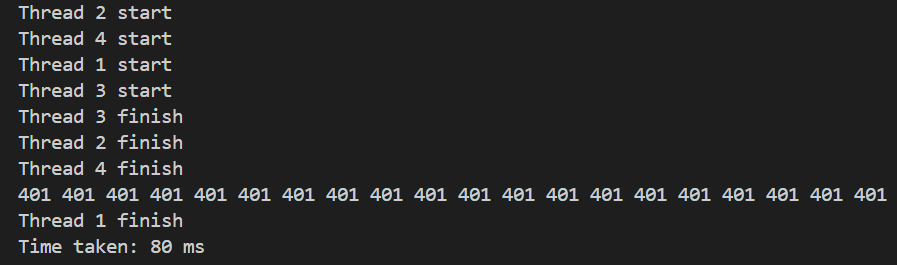
    }

}

**Етап 5. Тестування програми.**

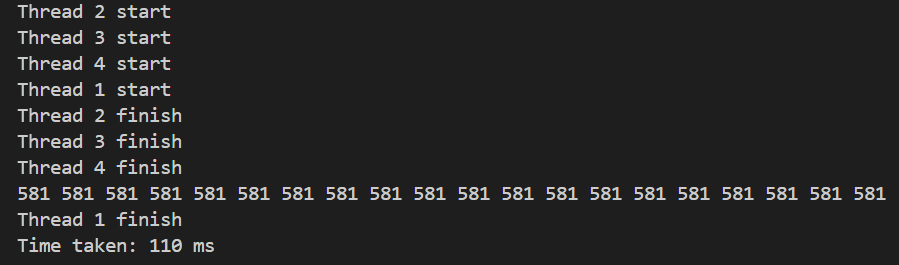
Оскільки формула обчислення міститься функцію max, доцільно буде порівняти результати програми, коли всі дані заповнені одиницями з тими, де є не одиничні елементи.

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 MD[1][2] = 10, що впливає на значення максимуму. Отримані результати співпадають з теоретичними розрахунками, а отже, програма обчислює формулу правильно.



Мал. 3 Вивід програми, коли MD[1][2] = 10

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=1500, було встановлено, що:

* час виконання програми, коли вона розподіляється системою автоматично по доступним ядрам: 1,2 секунди;
* час при обмеженні виконання лише 1-м ядром: 4,1 секунди.

Отже, Кп = 4,1/1,2 = 3,42.

**Висновки:**

1. розроблено паралельний математичний алгоритм, який дозволяє виконати обчислення заданої формули і визначити, що спільними ресурсами є скаляри a, p, e;
2. розроблені алгоритми кожної з задач, визначені завдання синхронізації, до яких входять синхронізація по введенню даних, виконанню обчислення 3 і виведенню результату, а також чотири завдання взаємного виключення пов'язаних з перезаписом або копіюванням спільних ресурсів;
3. побудовано структурну схему їх взаємодії, де засобами синхронізації є бар’єри і критичні секції;
4. реалізовано багатопоточну програму на мові Java, максимально наближаючись до функціональності паралельної обробки OpenMP, використовуючи ExecutorService;
5. проведено тестування, яке показало ефективність багатопоточної програми: при N = 1500 Кп = 3,42.