Kpi-best

МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМ. І.СІКОРСЬКОГО»

КАФЕДРА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ

**Лабораторна робота №4**

з дисципліни «Паралельне програмування»

на тему: «Мова Java. Монітори»

Виконав:

студент 3-го курсу

факультету ІОТ

групи ІО-14

Лупащенко А. А.

Перевірив:

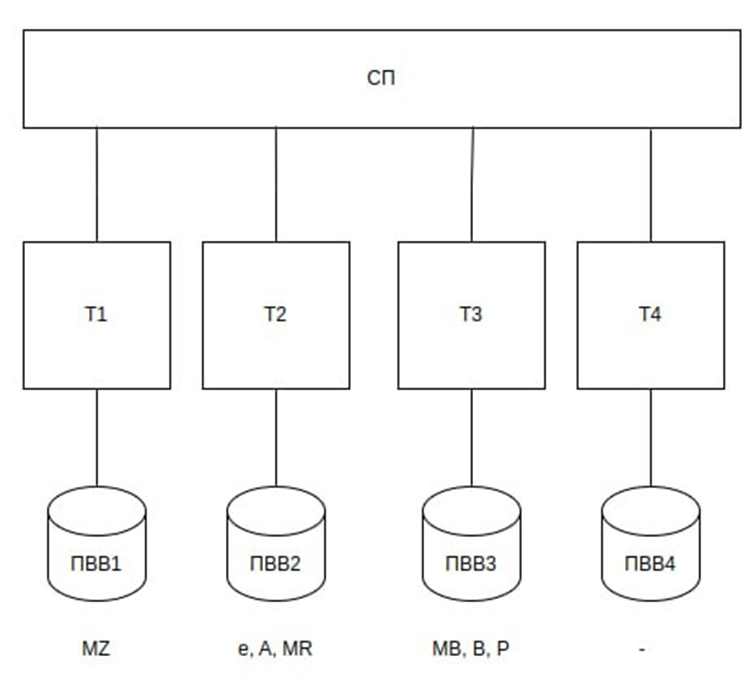
доц.

Корочкін О. В.

Київ 2023

**Варіант 27**

MO = MB\*(MC\*MM)\*d + min(Z)\*MC



MB, MC MO - Z, d, MM

**Етап 1. Паралельний алгоритм:**

1. аі = min(Zн) i = 1…4
2. a = mix(a,ai) СР:  a
3. MOH = MB\*(MC\*MMн)\*d + a\*MCн СР:  a, d

СР які змінюються: a

СР копіюємо: a, d

**Етап 2. Опис алгоритму потоків та точки синхронізації:**

T1:

1. Введення MB, MC
2. Сигнал Потокам T1,T3,T4 про введення MB, MC - - S(234,1)
3. Чекати на введення даних в потоках T2,T3,T4 - - W(234,1)
4. Копіювання d, d1 = d - - Критична ділянка
5. Обчислення а1 = min(Zн) - - Критична ділянка
6. Обчислення а = min(a, а1) - - Критична ділянка
7. Надсилаємо сигнал про завершення обчислень а – S(234,2)
8. Чекаємо на завершення обчислень а у інших потоках - W(234,2)
9. Копіювання a, а1 = a - - Критична ділянка
10. Обчислення MOH = MB\*(MC\*MMн)\*d + a\*MCн
11. Сигнал T2 про завершення обчислень - - S(2,3)

T2:

1. Чекати на введення даних в потоках T1,T3,T4 - - W(134,1)
2. Копіювання d, d2 = d - - Критична ділянка
3. Обчислення а2 = min(Zн) - - Критична ділянка
4. Обчислення а = min(a, а2) - - Критична ділянка
5. Надсилаємо сигнал про завершення обчислень а – S(134,1)
6. Чекаємо на завершення обчислень а у інших потоках - W(134,2)
7. Копіювання a, а2 = a - - Критична ділянка
8. Обчислення MOH = MB\*(MC\*MMн)\*d + a\*MCн
9. Чекаємо на завершення обчислень в T1,T3,T4 - -W(134,3)
10. Виведення результату MO

T3:

1. Чекати на введення даних в потоках T1,T2,T4 - - W(124,1)
2. Копіювання d, d3 = d - - Критична ділянка
3. Обчислення а3 = min(Zн) - - Критична ділянка
4. Обчислення а = min(a, а3) - - Критична ділянка
5. Надсилаємо сигнал про завершення обчислень а – S(124,1)
6. Чекаємо на завершення обчислень а у інших потоках - W(124,2)
7. Копіювання a, а3 = a - - Критична ділянка
8. Обчислення MOH = MB\*(MC\*MMн)\*d + a\*MCн
9. Сигнал T2 про завершення обчислень - - S(2,2)

T4:

1. Введення Z, d, MM
2. Сигнал Потокам T1,T2,T3 про введення Z, d, MM - - S(123,1)
3. Чекати на введення даних в потоках T1,T2,T3 - - W(123,1)
4. Копіювання d, d4 = d - - Критична ділянка
5. Обчислення а4 = min(Zн) - - Критична ділянка
6. Обчислення а = min(a, а4) - - Критична ділянка
7. Надсилаємо сигнал про завершення обчислень а – S(123,2)
8. Чекаємо на завершення обчислень а у інших потоках - W(123,2)
9. Копіювання a, а4 = a - - Критична ділянка
10. Обчислення MOH = MB\*(MC\*MMн)\*d + a\*MCн
11. Сигнал T2 про завершення обчислень - - S(2,3)

**Етап 3. Схема взаємодії потоків:**

**Етап 4. Код програми:**

import java.util.Arrays;

import java.util.concurrent.\*;

import java.util.concurrent.atomic.AtomicLong;

public class Lab2 {

private static final int n = 8;

private static final int P = 4;

private static final int H = n / P;

// Загальний об'єкт блокування для синхронізації потоків

public static final Object Lock = new Object();

// Бар'єр для синхронізації потоків

private static CyclicBarrier Bar1 = new CyclicBarrier(P);

// Семафор для взаємодії між T1, T2, T3, T4

private static Semaphore Sem1 = new Semaphore(0);

// Атомарна змінна для максимального значення

private static AtomicLong a = new AtomicLong(0);

// Ініціалізація даних (MD, MB, MA, C, E, d)

public static long[][] MD = new long[n][n];

public static long[][] MB = new long[n][n];

public static long[][] MA = new long[n][n];

public static long[] C = new long[n];

public static long[] E = new long[n];

public static long[] W = new long[n];

public static int d;

public static void main(String[] args) {

// Створення потоків

Thread t1 = new Thread(() -> T1());

Thread t2 = new Thread(() -> T2());

Thread t3 = new Thread(() -> T3());

Thread t4 = new Thread(() -> T4());

// Запуск потоків

long startTime = System.currentTimeMillis();

t1.start();

t2.start();

t3.start();

t4.start();

try {

// Очікування завершення роботи потоків

t1.join();

t2.join();

t3.join();

t4.join();

} catch (InterruptedException e) {

// Обробка випадку переривання: вивід стеку виклику в консоль

e.printStackTrace();

} finally {

long endTime = System.currentTimeMillis();

long executionTime = endTime - startTime;

System.out.println("Час виконання: " + executionTime + " мс");

}

}

private static void T1() {

int p = 1;

try {

System.out.println("T1: Початок роботи");

// Очікування введення у T2, T3, T4

Bar1.await();

// Критичний розділ - Копіювання та обчислення

int d1;

synchronized (Lock) {

d1 = d;

System.out.println("T1: Копіювання d");

}

long O1[] = scalarVectorMultiplication(C, MD, H \* (p - 1), H \* p);

long a1 = findMax(O1);

a.set(Math.max(a.get(), a1));

// Сигнал про завершення обчислень іншим потокам

Bar1.await();

// Критичний розділ - Копіювання

a1 = a.get();

// Обчислення Wh та запис до W

setValues(W, calcWh(p, H, a1, d1), H \* (p - 1), H \* p);

// Сигнал про завершення T3

Sem1.release();

} catch (InterruptedException | BrokenBarrierException e) {

e.printStackTrace();

} finally {

System.out.println("T1: Кінець роботи");

}

}

private static void T2() {

int p = 2;

try {

System.out.println("T2: Початок роботи");

// Введення MB, E

MB = generateMatrix(n, 2);

E = generateVector(3, n);

// Сигнал T1, T3, T4 про введення

System.out.println("T2: Дані завантажено");

Bar1.await();

// Критичний розділ - Копіювання та обчислення

int d2;

synchronized (Lock) {

d2 = d;

System.out.println("T2: Копіювання d");

}

long O2[] = scalarVectorMultiplication(C, MD, H \* (p - 1), H \* p);

long a2 = findMax(O2);

a.set(Math.max(a.get(), a2));

// Сигнал про завершення обчислень іншим потокам

Bar1.await();

// Критичний розділ - Копіювання

a2 = a.get();

// Обчислення Wh та запис до W

setValues(W, calcWh(p, H, a2, d2), H \* (p - 1), H \* p);

// Сигнал про завершення T3

Sem1.release();

} catch (InterruptedException | BrokenBarrierException e) {

e.printStackTrace();

} finally {

System.out.println("T2: Кінець роботи");

}

}

private static void T3() {

int p = 3;

try {

System.out.println("T3: Початок роботи");

// Введення C, MA

C = generateVector(2, n);

MA = generateMatrix(n, 4);

// Сигнал T1, T2, T4 про введення

System.out.println("T3: Дані завантажено");

Bar1.await();

// Критичний розділ - Копіювання та обчислення

int d3;

synchronized (Lock) {

d3 = d;

System.out.println("T3: Копіювання d");

}

long O3[] = scalarVectorMultiplication(C, MD, H \* (p - 1), H \* p);

long a3 = findMax(O3);

a.set(Math.max(a.get(), a3));

// Сигнал про завершення обчислень іншим потокам

Bar1.await();

// Критичний розділ - Копіювання

a3 = a.get();

// Обчислення Wh та запис до W

setValues(W, calcWh(p, H, a3, d3), H \* (p - 1), H \* p);

// Сигнал про завершення T1, T2, T4

Sem1.acquire(3);

System.out.println("W = " + Arrays.toString(W));

} catch (InterruptedException | BrokenBarrierException e) {

e.printStackTrace();

} finally {

System.out.println("T3: Кінець роботи");

}

}

private static void T4() {

int p = 4;

try {

System.out.println("T4: Початок роботи");

// Введення MD, d

MD = generateMatrix(n, 3);

d = 4;

// Сигнал T1, T2, T3 про введення

System.out.println("T4: Дані завантажено");

Bar1.await();

// Критичний розділ - Копіювання та обчислення

int d4;

synchronized (Lock) {

d4 = d;

System.out.println("T4: Копіювання d");

}

long O4[] = scalarVectorMultiplication(C, MD, H \* (p - 1), H \* p);

long a4 = findMax(O4);

a.set(Math.max(a.get(), a4));

// Сигнал про завершення обчислень іншим потокам

Bar1.await();

// Критичний розділ - Копіювання

a4 = a.get();

// Обчислення Wh та запис до W

setValues(W, calcWh(p, H, a4, d4), H \* (p - 1), H \* p);

// Сигнал про завершення T3

Sem1.release();

} catch (InterruptedException | BrokenBarrierException e) {

e.printStackTrace();

} finally {

System.out.println("T4: Кінець роботи");

}

}

// Метод для генерації матриці з одиницями по головній діагоналі

private static long[][] generateMatrix(int size, int values) {

long[][] matrix = new long[size][size];

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

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

if (i==j) {

matrix[i][j] = 1;

continue;

}

matrix[i][j] = values;

}

}

return matrix;

}

// Метод для генерації вектора з послідовними значеннями

private static long[] generateVector(int from, int count) {

int size = count;

long[] vector = new long[size];

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

vector[i] = i + from;

}

return vector;

}

// Метод для встановлення значень вектору з іншого вектору

private static void setValues(long[] obj, long[] values, int from, int to) {

for (int i = 0; i < to-from; i++) {

obj[i+from] = values[i];

}

}

// Метод для множення вектора на скаляр

public static long[] multiplyVectorByScalar(long[] vector, long scalar, int from, int to) {

int length = vector.length;

if (to == 0) {

to = length;

}

long[] result = new long[to-from];

for (int i = 0; i < to-from; i++) {

result[i] = vector[i+from] \* scalar;

}

return result;

}

// Метод для скалярного множення вектора на матрицю

private static long[] scalarVectorMultiplication(long[] vector, long[][] matrix, int from, int to) {

int rows = matrix.length;

int cols = matrix[0].length;

if (to == 0) {

to = cols;

}

if (vector.length != rows) {

throw new IllegalArgumentException("Несумісні розміри вектора та матриці");

}

long[] result = new long[to-from];

for (int i = 0; i < to-from; i++) {

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

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

}

}

return result;

}

// Метод для множення матриць

private static long[][] matrixMultiplication(long[][] matrix1, long[][] matrix2, int from, int to) {

int rows1 = matrix1.length;

int cols1 = matrix1[0].length;

int rows2 = matrix2.length;

if (cols1 != rows2) {

throw new IllegalArgumentException("Несумісні розміри матриці");

}

long[][] result = new long[rows1][to-from];

for (int i = 0; i < rows1; i++) {

for (int j = 0; j < to-from; j++) {

for (int k = 0; k < cols1; k++) {

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

}

}

}

return result;

}

// Метод для додавання векторів

private static long[] vectorAddition(long[] vector1, long[] vector2) {

int length = vector1.length;

if (length != vector2.length) {

throw new IllegalArgumentException("Несумісні векторні розміри");

}

long[] result = new long[length];

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

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

}

return result;

}

// Метод для пошуку максимального значення у векторі

private static long findMax(long[] vector) {

long max = vector[0];

for (int i = 1; i < vector.length; i++) {

if (vector[i] > max) {

max = vector[i];

}

}

return max;

}

// Метод обчислення Wh

private static long[] calcWh(int p, int H, long a, int d) {

//Calculating MA\*MB

long[][] result1 = matrixMultiplication(MA, MB, H\*(p-1), H\*p);

//Calculating E\*(MA\*MB)

long[] result2 = scalarVectorMultiplication(E, result1, 0, 0);

//Calculating E\*(MA\*MB)\*d

long[] result3 = multiplyVectorByScalar(result2, d, 0, 0);

//Calculating a\*C

long[] result4 = multiplyVectorByScalar(C, a, H\*(p-1), H\*p);

//Calculating a\*C + E\*(MA\*MB)\*d

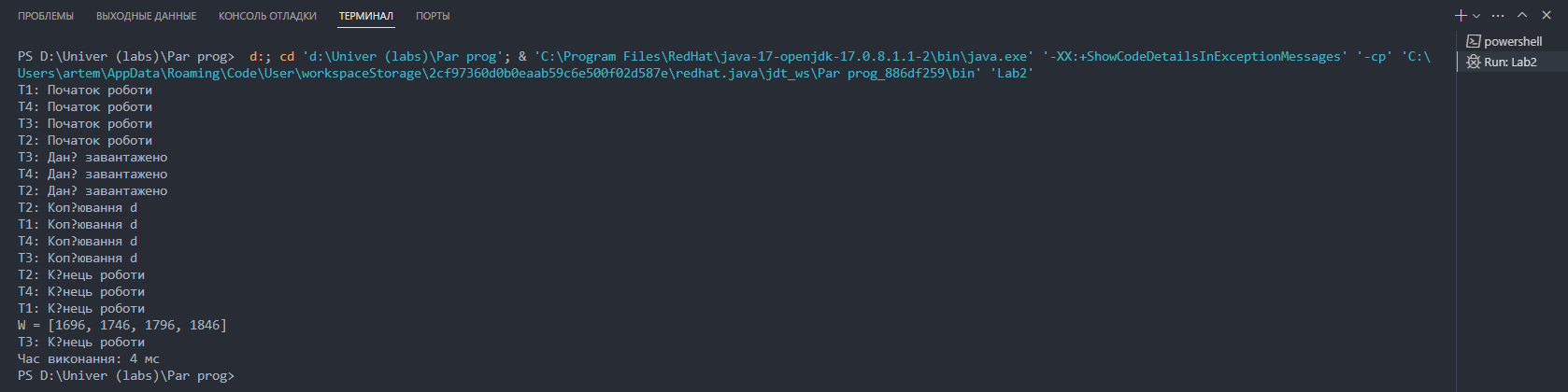
return vectorAddition(result4, result3);

}

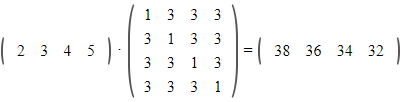
}

**Результати:**

Виконання для n = 4:

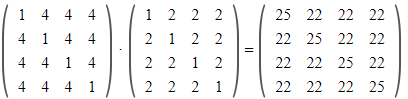


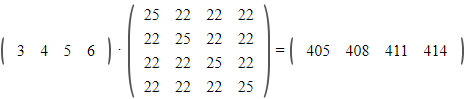
Перевірка значень на вірність:

C\*MD = 

max(C\*MD) = max(38 36 34 32) = 38

max(C\*MD)\*С = 

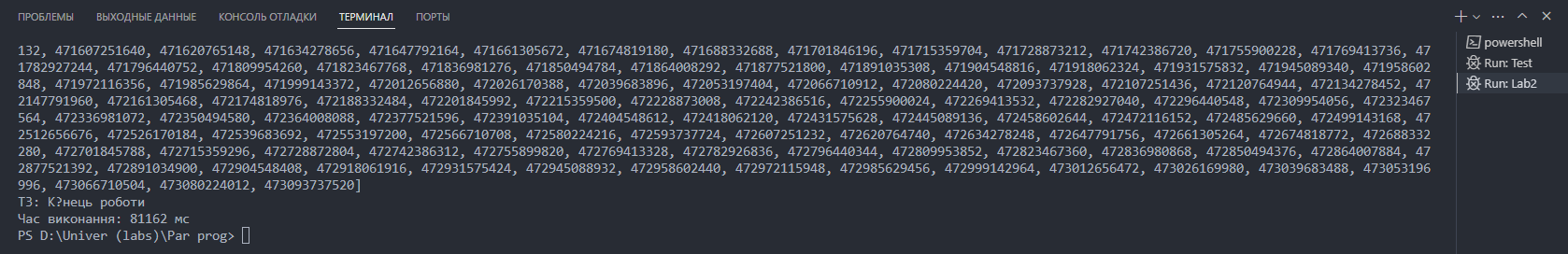
MA\*MB = 

E\*(MA\*MB) = 

E\*(MA\*MB)\*d = 

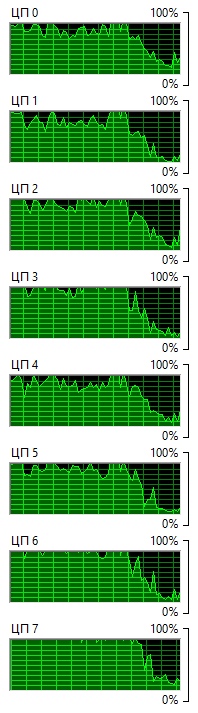
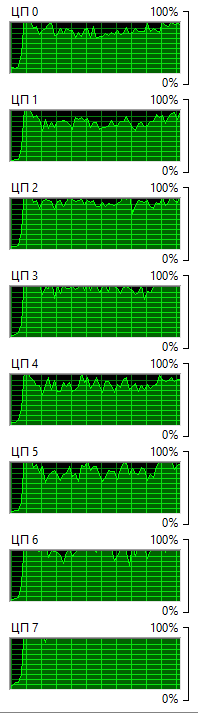
W = 

Виконання для n = 3000:



Навантаження ядер при n = 3000 становила 100%:

Дані наведені з використанням Resource Monitor ОС Windows



**Висновок:**

1. Розроблено програму, що використовує паралельні потоки для вирішення задачі з обчисленням. Java була обрана як мова програмування через наявність необхідних засобів для вирішення заданої роботи.
2. Під час розробки паралельного алгоритму виникло питання синхронізації потоків та захисту спільних ресурсів. Для цього були використані Semaphore для керування дозволами, критичні секції для забезпечення виключного доступу до частин коду, та AtomicInteger для безпечних операцій у багатопотоковому середовищі. Також використовувався CyclicBarrier для синхронізації групи потоків.
3. Розроблено алгоритми для кожного з чотирьох потоків, а також створено схему взаємодії потоків, відповідну розробленим алгоритмам. Ця схема гарантує відсутність дідлоків і правильну синхронізацію всіх потоків у програмі.
4. Під час тестування проблем не виявлено, що підтверджує точність розрахунків та успішне паралельне виконання завдань.