****

Міністерство освіти і науки України

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра інформатики і програмної інженерії

Лабораторна робота №8

з дисципліни

Технології паралельних обчислень

|  |  |  |
| --- | --- | --- |
| Виконав: |  | Перевірив: |
|  |  |  |
| студент групи ІП-13: |  | ст.викл. |
| Бабашев О. Д. |  | Дифучин А. Ю. |
|  |  |  |

Київ 2024

**Завдання до комп’ютерного практикуму 8 «Розробка алгоритмів для розподілених систем клієнт-серверної архітектури»**

1. Розробити веб-застосування клієнт-серверної архітектури, що реалізує алгоритм множення матриць або інший, який був Вами реалізований в рамках курсу «Технології паралельних обчислень», на стороні сервера з використанням паралельних обчислень. Розгляньте два варіанти реалізації 1) дані для обчислень знаходяться на сервері та 2) дані для обчислень знаходяться на клієнтській частині застосування.

2. Дослідити швидкість виконання запиту користувача при різних обсягах даних.

3. Порівняти реалізацію алгоритму в клієнт-серверній системі та в розподіленій системі з рівноправними процесорами.

**Хід роботи**

**Лістинг коду:**

**MatrixMultiplier.java**

package on\_client;

public class MatrixMultiplier implements Runnable {

private int[][] matrixA;

private int[][] matrixB;

private int[][] result;

private int startRow;

private int endRow;

public MatrixMultiplier(int[][] matrixA, int[][] matrixB, int[][] result, int startRow, int endRow) {

this.matrixA = matrixA;

this.matrixB = matrixB;

this.result = result;

this.startRow = startRow;

this.endRow = endRow;

}

@Override

public void run() {

int colsA = matrixA[0].length;

int colsB = matrixB[0].length;

for (int i = startRow; i < endRow; i++) {

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

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

result[i][j] += matrixA[i][k] \* matrixB[k][j];

}

}

}

}

}

**SequentialMatrixMultiplier.java**

package on\_client;

public class SequentialMatrixMultiplier {

private int[][] matrixA;

private int[][] matrixB;

private int[][] result;

public SequentialMatrixMultiplier(int[][] matrixA, int[][] matrixB) {

this.matrixA = matrixA;

this.matrixB = matrixB;

}

public int[][] multiplyMatrices() {

int rowsA = matrixA.length;

int colsB = matrixB[0].length;

int colsA = matrixA[0].length;

result = new int[rowsA][colsB];

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

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

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

result[i][j] += matrixA[i][k] \* matrixB[k][j];

}

}

}

return result;

}

}

**1) Дані для обчислень знаходяться на клієнтській частині застосування.**

**Client.java**

package on\_client;

import java.io.ObjectInputStream;

import java.io.ObjectOutputStream;

import java.net.Socket;

import java.util.Random;

public class Client {

public static void main(String[] args) {

try {

Socket socket = new Socket("localhost", 12345);

System.out.println("\nПідключено до сервера: " + socket + "\n");

ObjectOutputStream out = new ObjectOutputStream(socket.getOutputStream());

ObjectInputStream in = new ObjectInputStream(socket.getInputStream());

long startTime = System.nanoTime();

int SIZE = 500;

int[][] matrixA = generateMatrix(SIZE, SIZE);

int[][] matrixB = generateMatrix(SIZE, SIZE);

// Надсилання матриць на сервер

out.writeObject(matrixA);

out.writeObject(matrixB);

// Отримання результатів від сервера

int[][] parallelResult = (int[][]) in.readObject();

long endTime = System.nanoTime();

long duration = (endTime - startTime) / 1000000; // Конвертація часу в секунди

// Обчислення послідовного результату на клієнті

SequentialMatrixMultiplier sequentialMultiplier = new SequentialMatrixMultiplier(matrixA, matrixB);

int[][] sequentialResult = sequentialMultiplier.multiplyMatrices();

// Порівняння результатів

boolean isEqual = isEqual(sequentialResult, parallelResult);

System.out.println("Паралельний результат:");

printMatrix(parallelResult);

//System.out.println("Послідовний результат:");

//printMatrix(sequentialResult);

System.out.println("Результати корерктні: " + isEqual);

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

// Закриття з'єднання

socket.close();

} catch (Exception e) {

e.printStackTrace();

}

}

private static final Random random = new Random();

public static int[][] generateMatrix(int rows, int cols) {

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

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

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

matrix[i][j] = random.nextInt(2);

}

}

return matrix;

}

// Метод для порівняння двох матриць

public static boolean isEqual(int[][] matrixA, int[][] matrixB) {

if (matrixA.length != matrixB.length || matrixA[0].length != matrixB[0].length) {

return false;

}

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

for (int j = 0; j < matrixA[0].length; j++) {

if (matrixA[i][j] != matrixB[i][j]) {

return false;

}

}

}

return true;

}

public static void printMatrix(int[][] matrix) {

for (int[] row : matrix) {

for (int num : row) {

System.out.print(num + " ");

}

System.out.println();

}

}

}

**Server.java**

package on\_client;

import java.io.\*;

import java.net.\*;

public class Server {

public static void main(String[] args) {

try {

ServerSocket serverSocket = new ServerSocket(12345);

System.out.println("Сервер запущено. Очікування на з'єднаня...");

while (true) {

Socket clientSocket = serverSocket.accept();

System.out.println("Клієнт підключився: " + clientSocket);

ObjectOutputStream out = new ObjectOutputStream(clientSocket.getOutputStream());

ObjectInputStream in = new ObjectInputStream(clientSocket.getInputStream());

// Отримання матриць від клієнта

int[][] matrixA = (int[][]) in.readObject();

int[][] matrixB = (int[][]) in.readObject();

;

// Паралельне множення матриць

int numThreads = 16;

int[][] parallelResult = multiplyMatricesParallel(matrixA, matrixB, numThreads);

out.writeObject(parallelResult);

// Закриття з'єднання з клієнтом

clientSocket.close();

}

} catch (Exception e) {

e.printStackTrace();

}

}

// Метод для паралельного множення матриць за допомогою стрічкового алгоритму з вказаною кількістю потоків

public static int[][] multiplyMatricesParallel(int[][] matrixA, int[][] matrixB, int numThreads) {

int rowsA = matrixA.length;

int colsB = matrixB[0].length;

int[][] result = new int[rowsA][colsB];

// Обчислення кількості рядків матриці matrixA для кожного потоку

int rowsPerThread = (int) Math.ceil((double) rowsA / numThreads);

// Створення та запуск потоків

Thread[] threads = new Thread[numThreads];

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

int startRow = i \* rowsPerThread;

int endRow = Math.min((i + 1) \* rowsPerThread, rowsA);

threads[i] = new Thread(new MatrixMultiplier(matrixA, matrixB, result, startRow, endRow));

threads[i].start();

}

// Очікування завершення усіх потоків

try {

for (Thread thread : threads) {

thread.join();

}

} catch (InterruptedException e) {

e.printStackTrace();

}

return result;

}

}

**2) Дані для обчислень знаходяться на сервері.**

**Client.java**

package on\_server;

import java.io.ObjectInputStream;

import java.io.ObjectOutputStream;

import java.net.Socket;

public class Client {

public static void main(String[] args) {

try {

Socket socket = new Socket("localhost", 12345);

System.out.println("\nПідключено до сервера: " + socket + "\n");

ObjectOutputStream out = new ObjectOutputStream(socket.getOutputStream());

ObjectInputStream in = new ObjectInputStream(socket.getInputStream());

long startTime = System.nanoTime(); // Початок вимірювання часу

// Отримання результатів від сервера

int[][] parallelResult = (int[][]) in.readObject();

// Кінець вимірювання часу

long endTime = System.nanoTime();

long duration = (endTime - startTime) / 1000000; // в мілісекунди

// Виведення результатів

System.out.println("Паралельний результат:");

printMatrix(parallelResult);

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

// Закриття з'єднання

socket.close();

} catch (Exception e) {

e.printStackTrace();

}

}

public static boolean isEqual(int[][] matrixA, int[][] matrixB) {

if (matrixA.length != matrixB.length || matrixA[0].length != matrixB[0].length) {

return false;

}

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

for (int j = 0; j < matrixA[0].length; j++) {

if (matrixA[i][j] != matrixB[i][j]) {

return false;

}

}

}

return true;

}

public static void printMatrix(int[][] matrix) {

for (int[] row : matrix) {

for (int num : row) {

System.out.print(num + " ");

}

System.out.println();

}

}

}

**Server.java**

package on\_server;

import java.io.ObjectInputStream;

import java.io.ObjectOutputStream;

import java.net.ServerSocket;

import java.net.Socket;

import java.util.Random;

public class Server {

public static void main(String[] args) {

try {

ServerSocket serverSocket = new ServerSocket(12345);

System.out.println("Сервер запущено. Очікування на з'єднаня...");

while (true) {

Socket clientSocket = serverSocket.accept();

System.out.println("Клієнт підключився: " + clientSocket);

ObjectOutputStream out = new ObjectOutputStream(clientSocket.getOutputStream());

ObjectInputStream in = new ObjectInputStream(clientSocket.getInputStream());

int SIZE = 500;

int[][] matrixA = generateMatrix(SIZE, SIZE);

int[][] matrixB = generateMatrix(SIZE, SIZE);

// Паралельне множення матриць

int numThreads = 16;

int[][] parallelResult = multiplyMatricesParallel(matrixA, matrixB, numThreads);

// Відправка результатів клієнту

out.writeObject(parallelResult);

// Закриття з'єднання з клієнтом

clientSocket.close();

}

} catch (Exception e) {

e.printStackTrace();

}

}

private static final Random random = new Random();

public static int[][] generateMatrix(int rows, int cols) {

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

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

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

matrix[i][j] = random.nextInt(2);

}

}

return matrix;

}

// Метод для паралельного множення матриць за допомогою стрічкового алгоритму

public static int[][] multiplyMatricesParallel(int[][] matrixA, int[][] matrixB, int numThreads) {

int rowsA = matrixA.length;

int colsB = matrixB[0].length;

int[][] result = new int[rowsA][colsB];

// Обчислення кількості рядків матриці matrixA для кожного потоку

int rowsPerThread = (int) Math.ceil((double) rowsA / numThreads);

// Створення та запуск потоків

Thread[] threads = new Thread[numThreads];

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

int startRow = i \* rowsPerThread;

int endRow = Math.min((i + 1) \* rowsPerThread, rowsA);

threads[i] = new Thread(new MatrixMultiplier(matrixA, matrixB, result, startRow, endRow));

threads[i].start();

}

// Очікування завершення усіх потоків

try {

for (Thread thread : threads) {

thread.join();

}

} catch (InterruptedException e) {

e.printStackTrace();

}

return result;

}

}

**Результати виконання коду:**

При розробці застосунку для паралельного множення матриць на стороні серверу було використано стрічковий алгоритм. Перевіримо коректність роботи застосунку.

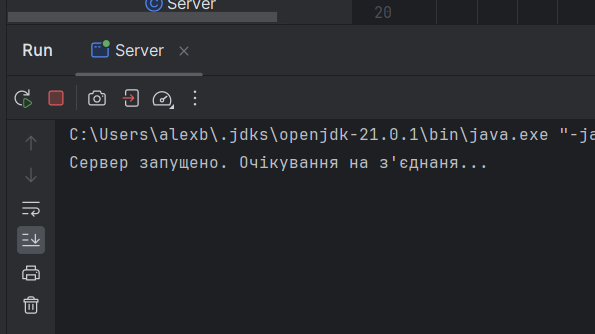


Рисунок 1.1 – Запуск сервера

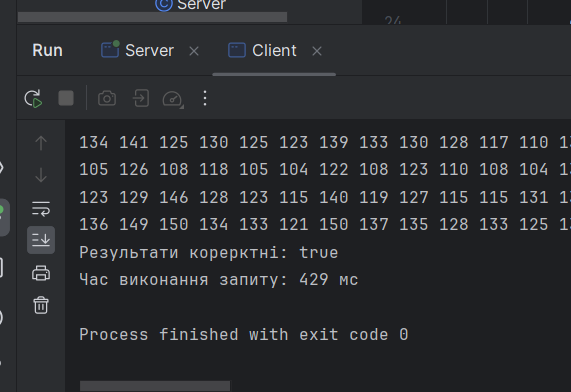


Рисунок 1.2 – Підключення клієнта, виконання множення та перевірка результату

Застосунок працює належним чином. Проведемо тестування обох розроблених реалізацій. Дослідимо швидкість виконання запиту користувача при різних обсягах даних. Маємо наступні результати дослідження.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| size  type | 500 | 1000 | 1500 | 2000 | 2500 |
| server | 341 | 3874 | 19689 | 47511 | 92968 |
| client | 720 | 4988 | 21421 | 49390 | 110499 |

Маємо цілком очікуваний результат. Реалізація із даними для обчислення на серверній частині працює швидше і розрив в часі із збільшенням розмірів матриць тільки збільшується. Це обумовлено тим, що в даній реалізації ми не витрачаємо час на відправку даних для обчислення на сервер, а одразу маємо їх на сервері і нам залишається тільки отримати результат обчислення на клієнті.

При порівнянні клієнт-серверної системи та розподіленої системи з рівноправними процесорами для множення матриць, виникають кілька ключових відмінностей. У клієнт-серверній архітектурі сервер виконує основні обчислення, а клієнти лише надсилають дані та отримують результати. Це спрощує вимоги до клієнтів і забезпечує централізоване управління ресурсами та безпекою, але створює єдину точку відмови. Натомість розподілена система децентралізована: кожен вузол має рівні права і виконує частину обчислень, що підвищує надійність і масштабованість, але ускладнює управління і синхронізацію.

**Висновок**

У ході виконання лабораторної роботи було розроблено веб-застосування клієнт-серверної архітектури множення матриць з паралельними обчисленнями стрічковим алгоритмом на сервері. Розглянуто два варіанти: дані знаходяться на сервері, і дані знаходяться на клієнті.

Дослідження швидкості виконання запитів показало, що в реалізації де дані для обчислення знаходяться на сервері працює швидше.

Порівняння клієнт-серверної системи та розподіленої системи з рівноправними процесорами показало, що клієнт-серверна архітектура простіша в управлінні та забезпечує централізовану обробку і зберігання даних, тоді як розподілена система більш масштабована і надійна, але складніша в координації.