Міністерство освіти і науки України

ДНІПРОВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ імені Олеся Гончара

Факультет фізики, електроніки та комп’ютерних систем

Кафедра електронних обчислювальних машин

Звіт

з лабораторної роботи №2

Виконав студент групи КІ-21-2 Браіла О.А.

Керівник: Скуратовський І. А.

м. Дніпро

2024

Завдання до лабораторної роботи:

Розробіть багатопотокову програму, яка виконує обчислення добутку матриць A (m×n) та B (n×k). Елементи cij матриці добутку = A×B обчислюються паралельно p(розмір пулу потоків) однотипними потоками. Якщо деякий потік вже обчислює елемент c{ij} матриці C, наступний потік, що приступає до обчислення, вибирає для розрахунку елемент c{i,j+1}, якщо j<k, і c{i+1,k}, якщо j=k. Виконавши обчислення елемента матриці-добутку потік перевіряє, чи немає елемента, який ще не розраховується. Якщо такий елемент є, то починає його розрахунок. В іншому випадку відправляє користувачу повідомлення про завершення своєї роботи та зупиняє своє виконання. Головний потік, отримавши повідомлення про завершення обчислень всіх потоків, виводить результат на екран. Вивід на екран матриці добутку виконайте форматовано.

Виконайте це завдання за допомогою одного з традиційних пулів потоків і за допомогою VirtualThreads. Порівняйте швидкодію а обох випадках.

Результат виконання програми:

A computer screen shot of a black screen

Description automatically generated

Рис.1. – Матриця 3х3 (1 спроба)

A black screen with white text

Description automatically generated

Рис.2. – Матриця 3х3 (2 спроба)

A screenshot of a computer

Description automatically generated

Рис.3. – Матриця 10х10 (1 спроба)

A screenshot of a computer

Description automatically generated

Рис.4. – Матриця 10х10 (2 спроба)

Тепер протестуємо програму та побудуємо таблицю значень:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| # | Matrix size | ThreadPool Time (ms) | Virtual Threads Time(ms) | Difference (Pool/Virtual) |
| 1 | 3x3 | 5 | 7 | 0,714 |
| 2 | 3x3 | 6 | 9 | 0,666 |
| 3 | 7x7 | 5 | 7 | 0,714 |
| 4 | 7x7 | 4 | 6 | 0,666 |
| 5 | 10x10 | 5 | 8 | 0,625 |
| 6 | 10x10 | 4 | 6 | 0,666 |
| 7 | 20x20 | 6 | 7 | 0,857 |
| 8 | 20x20 | 6 | 7 | 0,857 |
| 9 | 50x50 | 23 | 24 | 0,958 |
| 10 | 50x50 | 21 | 23 | 0,913 |
| 11 | 100x100 | 65 | 50 | 1,3 |
| 12 | 100x100 | 56 | 49 | 1,142 |
| 13 | 200x200 | 77 | 77 | 1 |
| 14 | 200x200 | 117 | 72 | 1,625 |
| 15 | 300x300 | 70 | 79 | 0,886 |
| 16 | 300x300 | 74 | 67 | 1,104 |

Код файлу MatrixMultiplier.java

package com.dnu.ffeks;  
  
import java.util.concurrent.\*;  
import java.util.\*;  
  
public class MatrixMultiplier {  
  
 // Розмірність матриць і кількість потоків  
 private static final int *M* = 300; // Кількість рядків у матриці A  
 private static final int *N* = 300; // Кількість стовпців у матриці A та рядків у матриці B  
 private static final int *K* = 300; // Кількість стовпців у матриці B  
 private static final int *P* = 4; // Кількість потоків у пулі потоків  
  
 public static void main(String[] args) throws InterruptedException, ExecutionException {  
 // Ініціалізація матриць  
 int[][] A = new int[*M*][*N*]; // Матриця A  
 int[][] B = new int[*N*][*K*]; // Матриця B  
 int[][] C1 = new int[*M*][*K*]; // Результат множення з ThreadPool  
 int[][] C2 = new int[*M*][*K*]; // Результат множення з VirtualThreads  
  
 // Заповнення матриць випадковими значеннями  
 *initializeMatrix*(A, 10); // Заповнюємо матрицю A числами від 0 до 9  
 *initializeMatrix*(B, 10); // Заповнюємо матрицю B числами від 0 до 9  
  
 // Виводимо матриці A і B  
 System.*out*.println("A-Matrix:");  
 *printMatrix*(A);  
 System.*out*.println("B-Matrix:");  
 *printMatrix*(B);  
  
 // Обчислюємо час роботи множення з використанням ThreadPool  
 long threadPoolTime = *measureTime*(() -> {  
 try {  
 *multiplyWithThreadPool*(A, B, C1);  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 });  
  
 // Обчислюємо час роботи множення з використанням VirtualThreads  
 long virtualThreadsTime = *measureTime*(() -> {  
 try {  
 *multiplyWithVirtualThreads*(A, B, C2);  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 });  
  
 // Виводимо результуючі матриці  
 System.*out*.println("Matrix C (ThreadPool):");  
 *printMatrix*(C1);  
 System.*out*.println("Matrix C (VirtualThreads):");  
 *printMatrix*(C2);  
  
 // Виводимо час виконання обчислень  
 System.*out*.printf("Time with ThreadPool: %d ms%n", threadPoolTime);  
 System.*out*.printf("Time with VirtualThreads: %d ms%n", virtualThreadsTime);  
 }  
  
 // Метод для ініціалізації матриці випадковими значеннями  
 private static void initializeMatrix(int[][] matrix, int maxValue) {  
 Random rand = new Random(); // Генератор випадкових чисел  
 for (int i = 0; i < matrix.length; i++) {  
 for (int j = 0; j < matrix[i].length; j++) {  
 matrix[i][j] = rand.nextInt(maxValue); // Генерація числа від 0 до maxValue - 1  
 }  
 }  
 }  
  
 // Метод для вимірювання часу виконання задачі  
 private static long measureTime(Runnable task) {  
 long start = System.*nanoTime*(); // Початок часу  
 task.run(); // Виконання задачі  
 long end = System.*nanoTime*(); // Кінець часу  
 return TimeUnit.*NANOSECONDS*.toMillis(end - start); // Повертаємо час у мілісекундах  
 }  
  
 // Множення матриць за допомогою ThreadPool  
 private static void multiplyWithThreadPool(int[][] A, int[][] B, int[][] C) throws InterruptedException {  
 ExecutorService executor = Executors.*newFixedThreadPool*(*P*); // Створення пулу потоків  
 ConcurrentLinkedQueue<int[]> taskQueue = new ConcurrentLinkedQueue<>(); // Черга завдань  
  
 // Додавання завдань у чергу  
 for (int i = 0; i < *M*; i++) {  
 for (int j = 0; j < *K*; j++) {  
 taskQueue.add(new int[]{i, j}); // Задача: обчислити елемент C[i][j]  
 }  
 }  
  
 CountDownLatch latch = new CountDownLatch(*M* \* *K*); // Синхронізація завершення завдань  
  
 // Створення потоків для обробки завдань  
 for (int i = 0; i < *P*; i++) {  
 executor.submit(() -> {  
 while (!taskQueue.isEmpty()) {  
 int[] task = taskQueue.poll(); // Витягуємо задачу з черги  
 if (task != null) {  
 int row = task[0]; // Рядок у матриці C  
 int col = task[1]; // Стовпець у матриці C  
 C[row][col] = 0; // Ініціалізація елемента C[row][col]  
 for (int t = 0; t < *N*; t++) {  
 C[row][col] += A[row][t] \* B[t][col]; // Обчислення елемента  
 }  
 latch.countDown(); // Зменшуємо лічильник після завершення завдання  
 }  
 }  
 });  
 }  
  
 latch.await(); // Очікуємо завершення всіх завдань  
 executor.shutdown(); // Закриваємо пул потоків  
 }  
  
 // Множення матриць за допомогою VirtualThreads  
 private static void multiplyWithVirtualThreads(int[][] A, int[][] B, int[][] C) throws InterruptedException {  
 ConcurrentLinkedQueue<int[]> taskQueue = new ConcurrentLinkedQueue<>(); // Черга завдань  
  
 // Додавання завдань у чергу  
 for (int i = 0; i < *M*; i++) {  
 for (int j = 0; j < *K*; j++) {  
 taskQueue.add(new int[]{i, j}); // Задача: обчислити елемент C[i][j]  
 }  
 }  
  
 CountDownLatch latch = new CountDownLatch(*M* \* *K*); // Синхронізація завершення завдань  
 List<Thread> threads = new ArrayList<>(); // Список віртуальних потоків  
  
 // Створення віртуальних потоків  
 for (int i = 0; i < *P*; i++) {  
 threads.add(Thread.*ofVirtual*().start(() -> {  
 while (!taskQueue.isEmpty()) {  
 int[] task = taskQueue.poll(); // Витягуємо задачу з черги  
 if (task != null) {  
 int row = task[0]; // Рядок у матриці C  
 int col = task[1]; // Стовпець у матриці C  
 C[row][col] = 0; // Ініціалізація елемента C[row][col]  
 for (int t = 0; t < *N*; t++) {  
 C[row][col] += A[row][t] \* B[t][col]; // Обчислення елемента  
 }  
 latch.countDown(); // Зменшуємо лічильник після завершення завдання  
 }  
 }  
 }));  
 }  
  
 latch.await(); // Очікуємо завершення всіх завдань  
 for (Thread thread : threads) {  
 thread.join(); // Чекаємо завершення кожного віртуального потоку  
 }  
 }  
  
 // Метод для виведення матриці на екран  
 private static void printMatrix(int[][] matrix) {  
 for (int[] row : matrix) {  
 for (int val : row) {  
 System.*out*.printf("%4d", val); // Вивід кожного елемента з фіксованою шириною  
 }  
 System.*out*.println(); // Перехід на новий рядок  
 }  
 }  
}

Посилання на GIT:

<https://github.com/brailaalexx/ABP-lab2>