МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО» ФАКУЛЬТЕТ ІНФОРМАТИКИ ТА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ

Кафедра інформатики та програмної інженерії

Звіт

З компʼютерного практикуму № 8 з дисципліни

«Технології паралельних обчислень»

Тема: « Розробка алгоритмів для розподілених систем клієнт-серверної архітектури»

**Виконав(ла)** *ІП-15 Мєшков Андрій*

(шифр, прізвище, ім'я, по батькові)

**Перевірив** *Дифучина О. Ю.*

(шифр, прізвище, ім'я, по батькові)

Київ 2024

# ЗАВДАННЯ

1. Розробити веб-застосування клієнт-серверної архітектури, що реалізує алгоритм множення матриць (або інший обчислювальний алгоритм, який був Вами реалізований іншими методами розподілених обчислень в рамках курсу «Паралельні та розподілені обчислення») 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. 60 балів.

2. Дослідити швидкість виконання запиту користувача при різних обсягах даних. 20 балів.

3. Порівняти реалізацію алгоритму в клієнт-серверній системи та в розподіленій системі з рівноправними процесорами. 20 балів.

ХІД РОБОТИ

Лістинг коду:

Сервер

ServerApplication.java

package com.example.demo;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class ServerApplication {

public static void main(String[] args) {

SpringApplication.run(ServerApplication.class, args);

}

}

MatrixController.java

package com.example.demo;

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.annotation.*\**;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import java.util.concurrent.RecursiveTask;

import java.util.concurrent.ForkJoinPool;

@RestController

@RequestMapping("/matrix")

public class MatrixController {

private static final Logger logger = LoggerFactory.getLogger(MatrixController.class);

private static final int THRESHOLD = 64; *// Threshold for parallel computation*

@PostMapping("/multiply-server")

public ResponseEntity<int[][]> multiplyServer(@RequestParam int size) {

logger.info("Received multiply-server request with size: {}", size);

int[][] A = generateMatrix(size);

int[][] B = generateMatrix(size);

*try* {

int[][] result = multiplyMatrices(A, B, size);

*return* ResponseEntity.ok(result);

} *catch* (Exception e) {

logger.error("Error during matrix multiplication", e);

*return* ResponseEntity.status(HttpStatus.INTERNAL\_SERVER\_ERROR).build();

}

}

@PostMapping("/multiply-client")

public ResponseEntity<int[][]> multiplyClient(@RequestBody int[][][] matrices) {

logger.info("Received multiply-client request");

*if* (matrices.length != 2) {

*throw* *new* IllegalArgumentException("Expected two matrices");

}

int[][] A = matrices[0];

int[][] B = matrices[1];

int size = A.length;

*try* {

int[][] result = multiplyMatrices(A, B, size);

*return* ResponseEntity.ok(result);

} *catch* (Exception e) {

logger.error("Error during matrix multiplication", e);

*return* ResponseEntity.status(HttpStatus.INTERNAL\_SERVER\_ERROR).build();

}

}

private int[][] generateMatrix(int size) {

int[][] matrix = *new* int[size][size];

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

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

matrix[i][j] = 1; *// Fill with 1s for simplicity*

}

}

*return* matrix;

}

private int[][] multiplyMatrices(int[][] A, int[][] B, int size) {

int[][] C = *new* int[size][size];

ForkJoinPool pool = *new* ForkJoinPool();

pool.invoke(*new* MatrixMultiplicationTask(A, B, C, 0, size, 0, size));

*return* C;

}

private static class MatrixMultiplicationTask extends RecursiveTask<Void> {

int[][] A, B, C;

int rowStart, rowEnd, colStart, colEnd;

MatrixMultiplicationTask(int[][] A, int[][] B, int[][] C, int rowStart, int rowEnd, int colStart, int colEnd) {

*this*.A = A;

*this*.B = B;

*this*.C = C;

*this*.rowStart = rowStart;

*this*.rowEnd = rowEnd;

*this*.colStart = colStart;

*this*.colEnd = colEnd;

}

@Override

protected Void compute() {

*if* ((rowEnd - rowStart) <= THRESHOLD || (colEnd - colStart) <= THRESHOLD) {

*for* (int i = rowStart; i < rowEnd; i++) {

*for* (int j = colStart; j < colEnd; j++) {

*for* (int k = 0; k < A.length; k++) {

C[i][j] += A[i][k] \* B[k][j];

}

}

}

*return* null;

} *else* {

int rowMid = (rowStart + rowEnd) / 2;

int colMid = (colStart + colEnd) / 2;

invokeAll(

*new* MatrixMultiplicationTask(A, B, C, rowStart, rowMid, colStart, colMid),

*new* MatrixMultiplicationTask(A, B, C, rowStart, rowMid, colMid, colEnd),

*new* MatrixMultiplicationTask(A, B, C, rowMid, rowEnd, colStart, colMid),

*new* MatrixMultiplicationTask(A, B, C, rowMid, rowEnd, colMid, colEnd)

);

*return* null;

}

}

}

}

Клієнт

ClientApplication.java

package com.example.client;

import org.springframework.boot.CommandLineRunner;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.web.reactive.function.client.WebClient;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

@SpringBootApplication

public class ClientApplication implements CommandLineRunner {

private static final Logger logger = LoggerFactory.getLogger(ClientApplication.class);

private final WebClient webClient;

public ClientApplication(WebClient.Builder webClientBuilder) {

*this*.webClient = webClientBuilder.baseUrl("http://localhost:8080/matrix").build();

}

public static void main(String[] args) {

SpringApplication.run(ClientApplication.class, args);

}

@Override

public void run(String... args) throws Exception {

int size = 3000; *// Example matrix size*

testServerComputation(size);

testClientComputation(size);

}

private void testServerComputation(int size) {

long startTime = System.currentTimeMillis();

*try* {

webClient.post()

.uri(uriBuilder -> uriBuilder

.path("/multiply-server")

.queryParam("size", size)

.build())

.retrieve()

.bodyToMono(int[][].class)

.doOnSuccess(result -> {

long endTime = System.currentTimeMillis();

logger.info("Server computation time: {} ms", (endTime - startTime));

logger.info("Result received from server:");

})

.block(); *// Block to wait for the Mono to complete*

} *catch* (Exception e) {

logger.error("Error during server computation", e);

}

}

private void testClientComputation(int size) {

int[][] A = generateMatrix(size);

int[][] B = generateMatrix(size);

long startTime = System.currentTimeMillis();

*try* {

webClient.post()

.uri("/multiply-client")

.bodyValue(*new* int[][][]{A, B})

.retrieve()

.bodyToMono(int[][].class)

.doOnSuccess(result -> {

long endTime = System.currentTimeMillis();

logger.info("Client computation time: {} ms", (endTime - startTime));

logger.info("Result received from client computation:");

})

.block(); *// Block to wait for the Mono to complete*

} *catch* (Exception e) {

logger.error("Error during client computation", e);

}

}

private int[][] generateMatrix(int size) {

int[][] matrix = *new* int[size][size];

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

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

matrix[i][j] = 1;

}

}

*return* matrix;

}

}

**Результат:**

В таблиці зображено порівняння алгоритмів.

|  |  |  |  |
| --- | --- | --- | --- |
|  | На сервері, с | Передається, с | Прискорення |
| 500 | 3,89 | 4,7692 | 1,226015424 |
| 1000 | 12,2485 | 15,175 | 1,238927216 |
| 2000 | 39,3357 | 49,9362 | 1,269488022 |
| 3000 | 60,4618 | 84,4053 | 1,396010373 |

# ВИСНОВКИ

В результаті роботи над комп’ютерним практикумом було розроблено програму, що реалізує паралельне множення матриць у клієнт-серверній архітектурі.