МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Комп’ютерний практикум №8

з курсу «Технології паралельних обчислень»

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

Викладач:

Дифучина О.Ю.

Виконала:

студентка 3 курсу

групи ІП-14 ФІОТ

Радзівіло Валерія

Київ-2023

**Завдання:**

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.

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

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

**Виконання завдання:**

**Завдання 1**

Цей проект використовує декілька технологій для розробки веб-додатку, який виконує операції з матрицями. Ось основні технології, які використовуються: Java: Основна мова програмування для серверної частини додатку. Використовується для написання бізнес-логіки, включаючи алгоритми обчислення матриць. Spring Boot: Фреймворк для створення веб-додатків на Java. Використовується для створення RESTful API, які взаємодіють з клієнтською частиною додатку. Maven: Інструмент для керування проектами на Java, який використовується для збірки проекту, керування залежностями та іншого. JavaScript: Мова програмування, яка використовується для клієнтської частини додатку. React: JavaScript-фреймворк для створення інтерфейсів користувача. Використовується для створення клієнтської частини додатку. NPM: Менеджер пакетів для JavaScript, який використовується для керування залежностями клієнтської частини додатку.

В цьому проекті використовується алгоритм Fox для множення матриць, який реалізований на Java. Веб-сервіс, створений за допомогою Spring Boot, надає API для взаємодії з цим алгоритмом. Клієнтська частина додатку, створена на React, використовує ці API для виконання операцій з матрицями.

**Реалізація №1 - дані для обчислень знаходяться на сервері**

@GetMapping("/{size}")  
public List<int[][]> showMatrices(@PathVariable String size) {  
 int s = Integer.*parseInt*(size);  
 Matrix matrix1 = new Matrix(s, s);  
 Matrix matrix2 = new Matrix(s, s);  
 matrix1.generateRandomMatrix();  
 matrix2.generateRandomMatrix();  
 return Arrays.*asList*(matrix1.matrix, matrix2.matrix);  
}

Метод «showMatrices», який використовується в контролері Spring Boot для обробки HTTP GET запитів на шлях «/{size}». «@PathVariable String size» означає, що значення «size» береться з частини URL, яка відповідає «{size}».

В методі «showMatrices» виконуються наступні дії: Значення «size», отримане з URL, перетворюється в ціле число «s» за допомогою «Integer.parseInt(size)». Створюються дві матриці «matrix1» та «matrix2» розміром «s x s» за допомогою конструктора класу «Matrix». Для обох матриць викликається метод «generateRandomMatrix()», який заповнює матриці випадковими числами. Метод повертає список цих двох матриць за допомогою «Arrays.asList(matrix1.matrix, matrix2.matrix)».

Отже, цей метод використовується для генерації двох випадкових матриць заданого розміру та відправки їх у відповідь на HTTP GET запит.

Файл `index.html` використовує React для створення інтерактивного веб-інтерфейсу. Ось основні елементи UI: Кнопка "Get Matrices from Java": Коли користувач натискає цю кнопку, виконується запит до сервера за адресою `http://127.0.0.1:8080/` з поточним розміром матриці. Отримані матриці відображаються на сторінці.

<button onClick={() => {  
 fetch('http://127.0.0.1:8080/' + matrixSize.valueOf())  
 .then(response => response.json())  
 .then(data => {  
 setMatrices(data);  
 setDataSourceInput(false);  
 })  
}}>Get Matrices from Java  
</button>

Поля вводу "Size" та "Thread count": Користувач може ввести розмір матриці та кількість потоків для обчислень.

<label>  
 Size:  
 <input type="number" name="size" min="1" max="500" defaultValue={matrixSize}  
 onChange={(e) => {  
 *console*.log(e.target.value);  
 setMatrixSize(e.target.value);  
 }}/>  
 </label>  
 <br/> <label>  
 Thread count:  
 <input type="number" name="size" min="1" max="10" defaultValue={threadCount}  
 onChange={(e) => {  
 *console*.log("New thread count : " + e.target.value);  
 setThreadCount(e.target.value);  
 }}/>  
</label>  
 <br/>

Відображення матриць: Якщо користувач не вводить матриці вручну, відображаються матриці, отримані з сервера. Кожна матриця відображається в таблиці.

{matrices.length > 0 && !dataSourceInput &&  
 matrices.map((matrix, matrixIndex) => (  
 <div key={matrixIndex}>  
 <h2>Matrix {matrixIndex + 1}</h2>  
 <table>  
 <tbody>  
 {matrix.map((row, rowIndex) => (  
 <tr key={rowIndex}>  
 {row.map((item, itemIndex) => (  
 <td key={itemIndex}>{item}</td>  
 ))}  
 </tr>  
 ))}  
 </tbody>  
 </table>  
 </div>  
 ))  
}

Кнопка "Multiply Matrices": Коли користувач натискає цю кнопку, виконується множення матриць. Результат відображається на сторінці.

{!dataSourceInput && <button onClick={  
 () => {  
 *console*.log("Multiply Matrices")  
  
 const payload = new MatrixPayload(matrices[0], matrices[1], threadCount.valueOf());  
 fetch('http://127.0.0.1:8080/multiply', {  
 method: 'POST',  
 headers: {  
 'Content-Type': 'application/json',  
 },  
 body: *JSON*.stringify(payload),  
  
 })  
 .then(response => {  
 if (!response.ok) {  
 throw new Error("HTTP error " + response.status);  
 }  
 return response.json();  
 })  
 .then(data => {  
 // output matrix result data on the screen  
 setResultMatrix(data);  
  
 *console*.log(data);  
 });  
  
 }  
}>Multiply Matrices  
</button>}

Відображення результату: Результат множення матриць відображається в таблиці під кнопкою "Multiply Matrices".

{resultMatrix.length > 0 && (  
 <div>  
 <h2>Result Matrix</h2>  
 <table>  
 <tbody>  
 {resultMatrix.map((row, rowIndex) => (  
 <tr key={rowIndex}>  
 {row.map((item, itemIndex) => (  
 <td key={itemIndex}>{item}</td>  
 ))}  
 </tr>  
 ))}  
 </tbody>  
 </table>  
 </div>  
)}

Цей інтерфейс дозволяє користувачу взаємодіяти з сервером, отримувати матриці, встановлювати розмір матриці та кількість потоків, а також отримувати результат множення матриць.

Алгоритм Фокса взятий з другої лабораторної роботи без вагомих змін. Ось так проходить множення матриць зі сторони Spring.

@PostMapping("/multiply")  
public int[][] multiplyMatrices(@RequestBody Map data) {  
 final MatrixPayload payload = MatrixPayload.*fromMap*(data);  
 Matrix firstMatrix = payload.getFirstMatrix();  
 Matrix secondMatrix = payload.getSecondMatrix();  
 FoxAlgorithm algorithm = new FoxAlgorithm(payload.getThreadCount(), firstMatrix, secondMatrix);  
 return algorithm.run(true);  
}  
  
@Setter  
@Getter  
public static class MatrixPayload implements Serializable {  
 private Matrix firstMatrix;  
 private Matrix secondMatrix;  
 private int threadCount;  
  
 MatrixPayload(Matrix firstMatrix, Matrix secondMatrix, int threadCount) {  
 this.firstMatrix = firstMatrix;  
 this.secondMatrix = secondMatrix;  
 this.threadCount = threadCount;  
 }  
  
 public static MatrixPayload fromMap(Map payload) {  
 return new MatrixPayload(  
 new Matrix((ArrayList) payload.get("firstMatrix")),  
 new Matrix((ArrayList) payload.get("secondMatrix")),  
 Integer.*parseInt*(payload.get("threadCount").toString()));  
  
 }  
  
}

**Висновок**

Виконуючи цю лабораторну роботу, було розроблено алгоритм паралельного множення матриць з використанням розподілених обчислень в MPI. Були використані методи колективного обміну повідомленнями, включаючи "один-до-одного","один-до-багатьох", "багато-до-одного" та "багато-до-багатьох".

В ході дослідження ефективності розподіленого обчислення алгоритму множення матриць було виявлено, що зі збільшенням розміру матриць та кількості вузлів, на яких здійснюється запуск програми, ефективність алгоритму зростає.

Порівняння ефективності алгоритму при використанні різних методів обміну повідомленнями показало, що метод "багато-до-багатьох" надає найкращі результати для даного завдання.

Ця робота дозволила глибше зрозуміти принципи роботи MPI та методи колективного обміну повідомленнями. Вона також показала важливість використання паралельних обчислень для оптимізації виконання великих обчислювальних завдань.

Посилання на код: <https://github.com/valeriia-radzivilo/parallel_lab7>