

**Національний Технічний Університет України КПІ**

Факультет Інформатики та Обчислювальної Техніки  
Кафедра інформатики та програмної інженерії

**Лабораторна робота №2**

З дисципліни «Технології паралельних обчислень»

Розробка паралельних алгоритмів множення матриць та дослідження їх ефективності

**Перевірила:**

Асистент

Дифучина Олександра Юріївна

Оцінка:\_\_\_\_\_\_\_\_\_\_\_**Виконав:**

Студент групи ІТ-92

Бондаренко Д.С.

Дата: \_\_\_\_\_\_\_\_\_\_\_\_\_

Завдання 1

Реалізуйте стрічковий алгоритм множення матриць. Результат множення записуйте в об’єкт класу Result.

У стрічковому алгоритмі кожний процес розраховує рядок результуючої матриці, кожна ітерація – один елемент рядка результуючої матриці.

Лістинг програми

Клас, який відповідає за множення та створення потоків

package StripedAlgorithm;  
import Shared.Result;  
import java.util.Arrays;  
  
public class StripedMultiplication {  
 public Result Multiply (double[][] firstMatrix, double[][] secondMatrix, int threads) throws InterruptedException {  
 Result result = new Result(firstMatrix.length, firstMatrix.length);  
 int rows = (int)Math.*ceil*((double)firstMatrix.length / threads);  
 StripedThread[] multiplyThreads = new StripedThread[threads];  
 for (int i = 0; i < threads; i++) {  
 int to = Math.*min*(i \* rows + rows, firstMatrix.length);  
 double[][] rowsMatrix = Arrays.*copyOfRange*(firstMatrix, i \* rows, to);  
 multiplyThreads[i] = new StripedThread(  
 rowsMatrix,  
 secondMatrix,  
 result,  
 i,  
 rows  
 );  
 multiplyThreads[i].start();  
 }  
 for (int i = 0; i < threads; i++) {  
 multiplyThreads[i].join();  
 }  
 return result;  
 }  
}

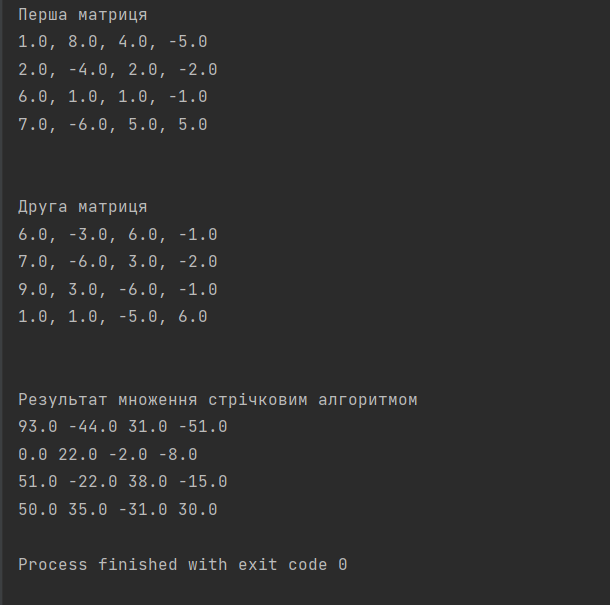
Клас-потік, у якому відбувається множення

package StripedAlgorithm;  
import Shared.Result;  
  
public class StripedThread extends Thread{  
 private final double[][] rowsMatrix;  
 private final double[][] secondMatrix;  
 private final Result result;  
 private final int rows;  
 private final int size;  
 private final int i;  
 private final int step;  
  
 public StripedThread (  
 double[][] firstMatrix,  
 double[][] secondMatrix,  
 Result resultMatrix,  
 int i,  
 int step  
 ) {  
 this.rowsMatrix = firstMatrix;  
 this.secondMatrix = secondMatrix;  
 this.result = resultMatrix;  
 this.i = i;  
 this.step = step;  
 this.rows = firstMatrix.length;  
 this.size = secondMatrix.length;  
 }  
  
 @Override  
 public void run() {  
 for (int j = 0; j < rows; j++) {  
 for (int k = 0; k < size; k++) {  
 double accumulator = 0;  
 for (int l = 0; l < size; l++) {  
 accumulator += this.rowsMatrix[j][l] \* this.secondMatrix[l][k];  
 }  
 this.result.SetValue(this.i \* this.step + j, k, accumulator);  
 }  
 }  
 }  
}

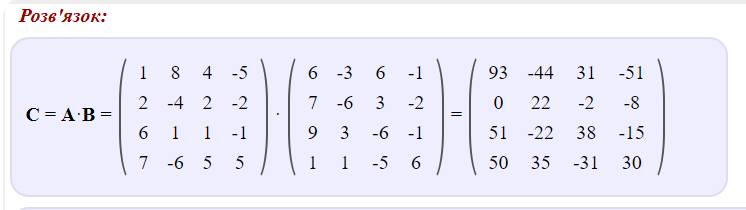
Клас Result

package Shared;  
  
public class Result {  
 public double[][] matrix;  
 int row;  
 int col;  
 public Result (int firstWidth, int secondLength) {  
 this.row = secondLength;  
 this.col = firstWidth;  
 this.matrix = new double[row][col];  
 for (int i = 0; i < row; i++) {  
 for (int j = 0; j < col; j++) {  
 this.matrix[i][j] = 0;  
 }  
 }  
 }  
 public void SetValue(int i, int j, double value){  
 this.matrix[i][j] = value;  
 }  
 public double GetValue(int i, int j){  
 return this.matrix[i][j];  
 }  
  
 public void Print() {  
 for (int i = 0; i < this.row; i++) {  
 for (int j = 0; j < this.col; j++) {  
 System.*out*.print(this.matrix[i][j]);  
 System.*out*.print(' ');  
 }  
 System.*out*.println();  
 }  
 }  
}

Результат множення матриць 4х4 на 4 потоках. Вивід у консоль



Для доказу роботи програми використаємо сторонній онлайн сервіс множення матриць (<https://ua.onlinemschool.com/math/assistance/matrix/multiply/>) та переконаємося у тому, що алгоритм працює вірно:

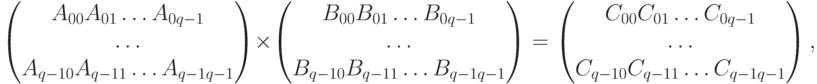


Як можна побачити, алгоритм працює вірно.

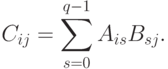
Завдання 2

Реалізуйте алгоритм Фокса множення матриць.

Алгоритм Фокса використовує блокову схема розбиття матриці. При такому способі поділу даних вихідні матриці А, B та результуюча матриця C представляються у вигляді наборів блоків. Далі передбачається, що всі матриці є квадратними розміру n×n, кількість блоків по горизонталі та вертикалі однаково і дорівнює q (тобто розмір всіх блоків дорівнює k×k, k=n/q). При такому поданні даних операція матричного множення матриць А та B у блочному вигляді може бути представлена ​​так:



Де кожен блок результуючої матриці визначається за формулою:



За основу паралельних обчислень для матричного множення при блоковому розділенні даних прийнятий підхід, при якому базові підзадачі відповідають за обчислення окремих блоків матриці C і при цьому в підзавданнях на кожній ітерації розрахунків розташовується лише по одному блоку вихідних матриць A і B.

Лістинг програми

Клас, який відповідає за множення та створення потоків

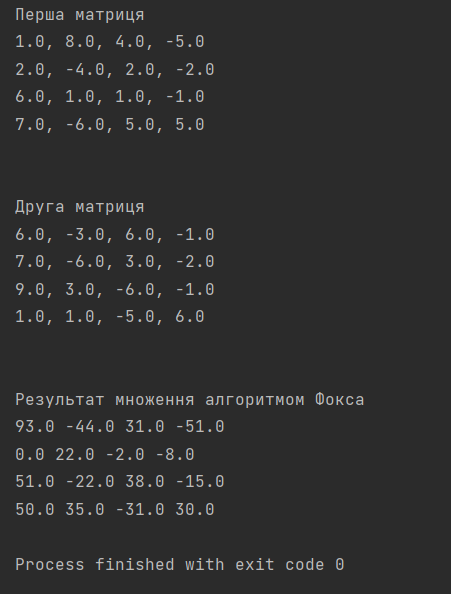
package FoxAlgorithm;  
import Shared.Result;  
import java.util.Arrays;  
  
public class FoxMultiplication {  
 public int rows, threadsSize;  
  
 public Result Multiply (double[][] firstMatrix, double[][] secondMatrix, int threads) throws InterruptedException {  
 this.threadsSize = (int) Math.*sqrt*(threads);  
 this.rows = Math.*min*(firstMatrix.length, firstMatrix.length/threadsSize);  
 double[][][][] blockedFirstMatrix = getBlockedMatrix(firstMatrix);  
 double[][][][] blockedSecondMatrix = getBlockedMatrix(secondMatrix);  
 FoxThread[][] multiplyThreads = new FoxThread[threadsSize][threadsSize];  
 Result result = new Result(firstMatrix.length, firstMatrix.length);  
 for (int stage = 0; stage < threadsSize; stage++ ) {  
 for (int i = 0; i < threadsSize; i++) {  
 for (int j = 0; j < threadsSize; j++) {  
 multiplyThreads[i][j] = new FoxThread(  
 blockedFirstMatrix[i][i + stage >= threadsSize ? i + stage - threadsSize : i + stage],  
 blockedSecondMatrix[i + stage >= threadsSize ? i + stage - threadsSize : i + stage][j],  
 result,  
 i,  
 j  
 );  
 multiplyThreads[i][j].start();  
 }  
 }  
 }  
 for (int i = 0; i < threadsSize; i++) {  
 for (int j = 0; j < threadsSize; j++) {  
 multiplyThreads[i][j].join();  
 }  
 }  
 return result;  
 }  
  
 private double[][][][] getBlockedMatrix(double[][] matrix) {  
 double[][][][] blockedMatrix = new double[threadsSize][threadsSize][rows][rows];  
 for (int i = 0; i < threadsSize; i++) {  
 for (int j = 0; j < threadsSize; j++) {  
 double[][] block = new double[rows][rows];  
 for (int k = 0; k < rows; k++) {  
 int to = i == threadsSize - 1 ? matrix[i \* rows + k].length : j \* rows + rows;  
 block[k] = Arrays.*copyOfRange*(matrix[i \* rows + k], j \* rows, to);  
 }  
 blockedMatrix[i][j] = block;  
 }  
 }  
 return blockedMatrix;  
 }  
}

Клас-потік, у якому відбувається множення

package FoxAlgorithm;  
import Shared.Result;  
  
public class FoxThread extends Thread {  
 private final double[][] firstBlock;  
 private final double[][] secondBlock;  
 private final Result result;  
 private final int i;  
 private final int j;  
 private final int rows;  
  
 public FoxThread (  
 double[][] blockedFirstMatrix,  
 double[][] blockedSecondMatrix,  
 Result result,  
 int i,  
 int j  
 ) {  
 this.firstBlock = blockedFirstMatrix;  
 this.secondBlock = blockedSecondMatrix;  
 this.result = result;  
 this.rows = blockedFirstMatrix.length;  
 this.i = i;  
 this.j = j;  
 }  
  
 @Override  
 public void run() {  
 for (int k = 0; k < rows; k++) {  
 for (int l = 0; l < rows; l++) {  
 double val = this.result.GetValue(this.i \* rows + k, this.j \* rows + l);  
 int accumulator = 0;  
 for (int m = 0; m < rows; m++) {  
 accumulator += this.firstBlock[k][m] \* this.secondBlock[m][l];  
 }  
 this.result.SetValue(this.i \* rows + k, this.j \* rows + l, val + accumulator);  
 }  
 }  
 }  
}

Лістинг класу Result можна подивитися у минулому завданні.

Результат множення матриць 4х4 на 4 потоках. Вивід у консоль



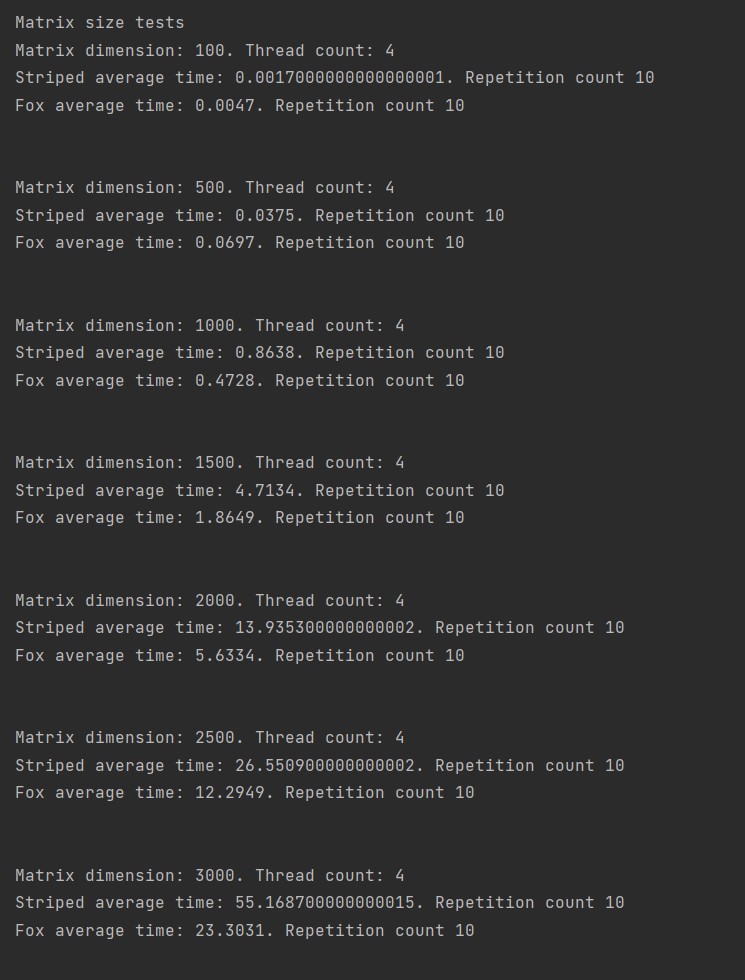
Результат множення такий самий як і у минулому завданні, а отже алгоритм працює вірно.

Завдання 3

Виконайте експерименти, варіюючи розмірність матриць, які перемножуються, для обох алгоритмів, та реєструючи час виконання алгоритму. Порівняйте результати дослідження ефективності обох алгоритмів.

Для кожного розміру матриць було виконано 10 замірів та взято середнє значення часу в секундах. Для замірів ми будемо використовувати матриці розмірів 100, 500, 1000, 1500, 2000, 2500, 3000. Всі заміри виконані на 4 потоках.

Отримаємо такі результати:



|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Розмір | 100 | 500 | 1000 | 1500 | 2000 | 2500 | 3000 |
| Стрічковий | 0,0017 | 0,0375 | 0,8638 | 4,7134 | 13,9353 | 26,5509 | 55,1687 |
| Фокс | 0,0047 | 0,0697 | 0,4728 | 1,8649 | 5,6334 | 12,2949 | 23,3031 |

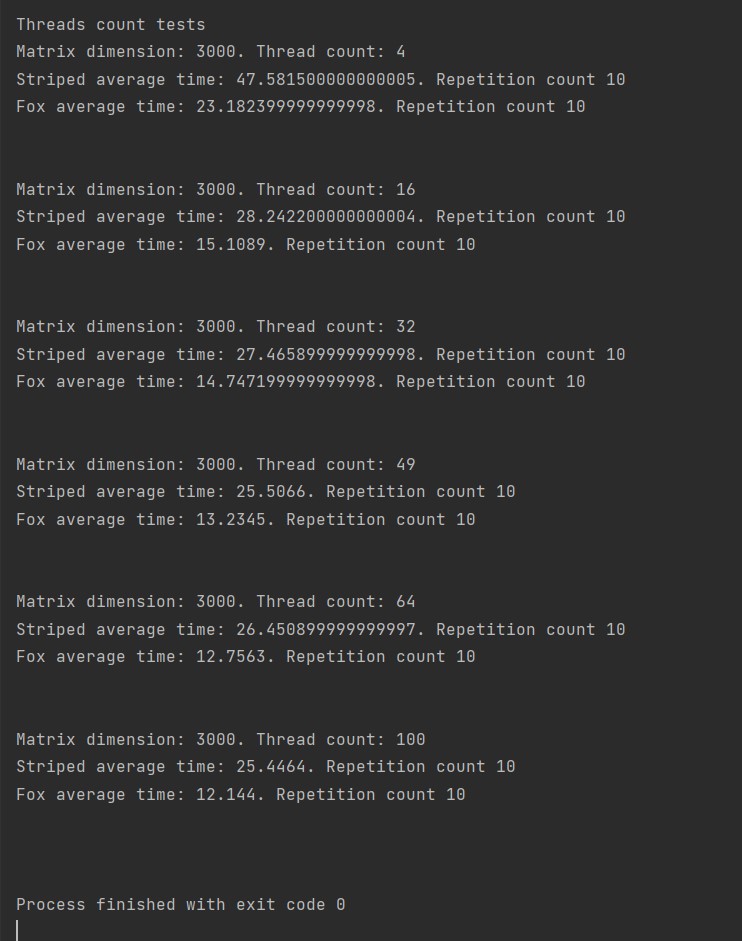
Як можна побачити із таблиці, стрічковий алгоритм показує себе краще при меншому розмірі матриць, ніж алгоритм Фокса. Оскільки рахунок йде на мілісекунди, то я гадаю, що виграш у продуктивності ми отримуємо через те, що у стрічковому алгоритмі не відбувається створення матриць матриць та комплексне розбиття на блоки. В загальному плані, алгоритм Фокса показав себе краще по швидкості виконання множення.

Завдання 4

Виконайте експерименти, варіюючи кількість потоків, що використовується для паралельного множення матриць, та реєструючи час виконання. Порівняйте результати дослідження ефективності обох алгоритмів.

Для кожної кількості потоків було виконано 10 замірів та взято середнє значення часу в секундах. Для замірів ми будемо використовувати такі кількості потоків, як 4, 16, 32, 49, 64, 100. Всі заміри виконані на матрицях розміром 3000х3000.

Отримаємо такі результати:



|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Потоки | 4 | 16 | 32 | 49 | 64 | 100 |
| Стрічковий | 47,5815 | 28,2422 | 27,4658 | 25,5066 | 26,4508 | 25,4464 |
| Фокс | 23,1823 | 15,1089 | 14,7471 | 13,2345 | 12,7563 | 12,144 |

Як можна побачити, зі збільшенням кількості потоків, алгоритми починали виконуватися швидше, а отже і ставали ефективнішими. Найкраще себе все рівно показав алгоритм Фокса, який у всіх випадках працював швидше. Найбільший приріст ми отримали при переході із 4 на 16 потоків. При подальшому збільшуванні, час також зменшувався, але не на багато. Я гадаю, що якщо комп’ютер буде використовуватися лише для обрахунку множення матриць, то варто брати найбільш можливу кількість потоків, якщо ж на комп’ютері має виконуватися також інша робота окрім множення матриць, то треба балансувати між навантаженням на процесор та часом. На мою думку, треба брати 16 потоків, оскільки виграш у декілька секунд не завжди буде пріоритетним ніж інша робота, яку може виконувати процесор разом із множенням.

Висновок

У даній лабораторній роботі ми реалізували два алгоритми множення матриць – стрічковий та Фокса. Обидва алгоритми потрібно використовувати в залежності від поставленої задачі – якщо елементів не багато, то швидшим буде більш простий стрічковий алгоритм, якщо ж елементів багато, то краще використовувати алгоритм Фокса, оскільки саме він буде швидше. Також ми порівняли алгоритми в залежності від розмірів матриці та від кількості потоків. Як вже наголошувалося, при збільшені кількості елементів буде працювати краще алгоритм Фокса. При збільшені кількості потоків алгоритми також починали загалом працювати швидше, але, на мою думку, виграш у декілька секунд не вартий того, що процесор не може виконувати більш ніяку роботу, окрім множення матриць. Варто також наголосити, що дані висновки та дослідження будуть актуальні лише для комп’ютера із такою конфігурацією, як:

Процесор: частота 3.6 ГГЦ; 8 фізичних та логічних ядер.

Оперативна пам’ять: 32 гб.

На інших конфігураціях комп’ютера, результати роботи алгоритмів можуть відрізнятися.

Повний лістинг програми можна знайти на GitHub: https://github.com/Agupnik/PDC/tree/master/PDCLab2/PDCLab2