JAVA 编程进阶上机报告

学　院　智能与计算学部

专　业　软件工程

班　级　三班

学　号　3018216158

姓　名　陈雅鹏

一、实验要求

• 编写矩阵随机生成类 MatrixGenerator 类，随机生成任意大小的矩阵，矩阵单元使用 double 存储。

• 使用串行方式实现矩阵乘法。

• 使用多线程方式实现矩阵乘法。

• 比较串行和并行两种方式使用的时间，利用第三次使用中使用过的 jvm 状态查看命令，分析产生时间差异的原因是什么。

二、源代码

MatrixGenerator

```
import java.util.Random;

/**
 * 工具类，生成随机矩阵
 * @author tju cyp
 */

public class MatrixGenerator {
    private static Random random = new Random();
    /**
     * 生成指定大小的随机矩阵,元素大小在[min,max]区间内
     * @param row
     * @param column
     * @param min
     * @param max
     * @return
     */
    public static double[][] getRandomMatrix(int row, int column, long
min, long max){
        double[][] matrix = new double[row][column];
        for (int i = 0 ; i < row ; i++){
            for (int j = 0 ; j < column ; j++){
                matrix[i][j] = min + random.nextDouble()*(max-min);
            }
        }
        return matrix;
    }
}
```

SerialMultiply

```java
import java.util.concurrent.CountDownLatch;

/**
 * 串行类
 */
public class SerialMultiply implements Runnable{
    private double[][] A;
    private double[][] B;
    private int row;
    private int column;
    private CountDownLatch latch;

    public SerialMultiply(double[][] a, double[][] b , CountDownLatch l
atch) {
        A = a;
        B = b;
        this.latch = latch;
        row = a.length;
        column = b[0].length;
    }

    @Override
    public void run() {
        double[][] C = new double[row][column];
        for (int i = 0 ; i < row ; i++){
            for (int j = 0 ; j < column ; j++){
                double res = 0;
                for (int k = 0 ; k < A[0].length ; k++){
                    res += A[i][k]*B[k][j];
                }
                C[i][j] = res;
            }
        }
        latch.countDown();
    }
}
```

ParallelMultiply

```java
import java.util.concurrent.CountDownLatch;

/**
 * 并行类
 */
public class ParallelMultiply implements Runnable{
    private double[][] A;
    private double[][] B;
    private int row;
    private int column;
    private int temp;
```

```
    private CountDownLatch latch;

    public ParallelMultiply(double[][] a, double[][] b, CountDownLatch
latch, int temp){
        A = a;
        B = b;
        this.latch = latch;
        this.temp = temp;
        row = a.length;
        column = b[0].length;
    }

    @Override
    public void run() {
        double[][] C = new double[row][column];
        for (int i = temp ; i < row ; i += 2){
            for (int j = 0 ; j < column ; j++){
                double res = 0;
                for (int k = 0 ; k < A[0].length ; k++){
                    res += A[i][k]*B[k][j];
                }
                C[i][j] = res;
            }
        }
        latch.countDown();
    }
}
```

Main

```
import java.util.Scanner;
import java.util.concurrent.CountDownLatch;

public class Main {
    public static void main(String[] args) throws InterruptedException
{
        int rowA,columnA,rowB,columnB;
        double[][] a,b;
        CountDownLatch latch1,latch2;
        while (true){
            System.out.println("输入两个矩阵的大小:");
            Scanner input = new Scanner(System.in);
            rowA = input.nextInt();
            columnA = input.nextInt();
            rowB = input.nextInt();
            columnB = input.nextInt();

            long totalTime1 = 0;
            long totalTime2 = 0;
```

```
            for (int i = 0; i < 10; i++){
                a = MatrixGenerator.getRandomMatrix(rowA,columnA,0,1000
0);
                b = MatrixGenerator.getRandomMatrix(rowB,columnB,0,1000
0);
                latch1 = new CountDownLatch(1);
                Thread thread = new Thread(new SerialMultiply(a,b,latch
1));
                long startTime1 = System.currentTimeMillis();
                thread.start();
                latch1.await();
                long endTime1 = System.currentTimeMillis();
                totalTime1 += endTime1 -startTime1;

                latch2 = new CountDownLatch(2);
                Thread thread1 = new Thread(new ParallelMultiply(a,b,la
tch2,0));
                Thread thread2 = new Thread(new ParallelMultiply(a,b,la
tch2,1));
                long startTime2 = System.currentTimeMillis();
                thread1.start();
                thread2.start();

                latch2.await();
                long endTime2 = System.currentTimeMillis();
                totalTime2 += endTime2 -startTime2;
            }

            System.out.println("串行执行平均耗时：" + totalTime1/10 + " m
s");
            System.out.println("并行执行平均耗时：" + totalTime2/10 + " m
s");
            System.out.println("----------------------------------------
--------");
        }
    }
}
```

三、设计思路

- MatrixGenerator

  工具类，包含静态方法 getRandomMatrix(int row, int column, long min, long max)

  生成一个指定大小和数据范围的随机矩阵

- SerialMultiply

  串行矩阵乘法计算类，使用最普通的矩阵乘法算法

- ParallelMultiply

  并行矩阵乘法计算类，将矩阵分成两部分，计数行和偶数行单独计算

- Main

  测试分析类，输入两个矩阵的规格，生成 10 次随机矩阵并分别使用串行和并行的方式相乘，输出 10 次的运行时间平均数

四、运行截图

```
输入两个矩阵的大小:
100 100
100 100
串行执行平均耗时: 2 ms
并行执行平均耗时: 2 ms
-----------------------------------------------
输入两个矩阵的大小:
100 200
200 100
串行执行平均耗时: 3 ms
并行执行平均耗时: 1 ms
-----------------------------------------------
输入两个矩阵的大小:
200 200
200 200
串行执行平均耗时: 11 ms
并行执行平均耗时: 6 ms
-----------------------------------------------
输入两个矩阵的大小:
200 400
400 200
串行执行平均耗时: 24 ms
并行执行平均耗时: 16 ms
-----------------------------------------------
```

```
输入两个矩阵的大小:
400 400
400 400
串行执行平均耗时：106 ms
并行执行平均耗时：63 ms
-----------------------------------------------
输入两个矩阵的大小:
200 500
500 200
串行执行平均耗时：31 ms
并行执行平均耗时：19 ms
-----------------------------------------------
输入两个矩阵的大小:
500 500
500 500
串行执行平均耗时：206 ms
并行执行平均耗时：125 ms
-----------------------------------------------
输入两个矩阵的大小:
500 800
800 500
串行执行平均耗时：441 ms
并行执行平均耗时：284 ms
-----------------------------------------------
```

```
输入两个矩阵的大小:
800 800
800 800
串行执行平均耗时：1470 ms
并行执行平均耗时：1145 ms
-----------------------------------------------
输入两个矩阵的大小:
800 1000
1000 800
串行执行平均耗时：2441 ms
并行执行平均耗时：1714 ms
-----------------------------------------------
输入两个矩阵的大小:
1000 1000
1000 1000
串行执行平均耗时：5825 ms
并行执行平均耗时：3864 ms
-----------------------------------------------
输入两个矩阵的大小:
1000 2000
2000 1000
串行执行平均耗时：16439 ms
并行执行平均耗时：9869 ms
-----------------------------------------------
```

五、分析

由运行结果分析看出，矩阵较小时，并行效率与串行效率相差不大甚至串行强于并行。这可能是由于计算矩阵乘法的开销与开启新线程的开销相差不大造成的，并不能体现出并行的优势。而当矩阵规模增大时，并行效率明显大于串行效率，运行时间几乎是串行的一半，这时由于总时间远远大于开启新线程所花费的时间，开启新线程所用的时间几乎可以忽略。这也印证了两个线程相比于一个线程计算的优势。