# 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 6班**

**学 号 3018216298**

**姓 名 米思成**

1. **实验要求**

第四次实验是使用多线程编程技术，编写矩阵乘法。

## 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。

1. **源代码**

**MatrixGenerator.java**

**public** **class** MatrixGenerator {

**public** **final** **static** **int** ***MAX*** = 10; //定义矩阵可能的最大维度

**public** **int** i = (**int**) (1 + (Math.*random*() \* ***MAX***));

**public** **int** j = (**int**) (1 + (Math.*random*() \* ***MAX***));

**public** **int** k = (**int**) (1 + (Math.*random*() \* ***MAX***));

**public** **double**[][] mx1 = **new** **double**[i][j];

**public** **double**[][] mx2 = **new** **double**[j][k];

**public** MatrixGenerator(){

//生成第一个矩阵

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

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

mx1[m][n] = (-2) + (Math.*random*() \* (3+2));

//保留两位小数

BigDecimal bg = **new** BigDecimal(mx1[m][n]);

mx1[m][n] = bg.setScale(2, BigDecimal.***ROUND\_HALF\_UP***).doubleValue();

}

}

//生成第二个矩阵

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

**for**(**int** n = 0; n<k; n++) {

mx2[m][n] = (-2) + (Math.*random*() \* (3+2));

//保留两位小数

BigDecimal bg = **new** BigDecimal(mx2[m][n]);

mx2[m][n] = bg.setScale(2, BigDecimal.***ROUND\_HALF\_UP***).doubleValue();

}

}

}

}

**Serial.java**

**public** **class** Serial {

**public** **static** **double**[][] SerialCalculate(**double**[][] mx1, **double**[][] mx2){

**int** i = mx1.length;

**int** j = mx2.length;

**int** k = mx2[0].length;

**double**[][] M = **new** **double**[i][k];

System.***out***.println("串行计算结果：");

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

**for**(**int** y = 0; y<k; y++) {

M[x][y] = 0;

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

M[x][y] += mx1[x][z] \* mx2[z][y];

}

//保留两位小数

BigDecimal bg = **new** BigDecimal(M[x][y]);

M[x][y] = bg.setScale(2, BigDecimal.***ROUND\_HALF\_UP***).doubleValue();

System.***out***.print(M[x][y] + " ");

}

System.***out***.println();

}

**return** M;

}

}

**Concurrency.java**

**public** **class** Concurrency {

**public** **static** **void** main(String[] args) **throws** InterruptedException {

//这里分为两个线程分别计算奇数行和偶数行的矩阵乘法

MatrixThread mt = **new** MatrixThread();

Thread t1 = **new** Thread(mt, "线程1");

Thread t2 = **new** Thread(mt, "线程2");

t1.start();

t2.start();

**double**[][] MC = mt.print();

//使用断言判断结果正确性

**double**[][] mx1 = mt.ReturnMx1();

**double**[][] mx2 = mt.ReturnMx2();

**double**[][] MS = Serial.*SerialCalculate*(mx1, mx2);

**assert** (*MxEqual*(MS,MC)):"经断言判断，串行计算结果与并发计算结果不同";

System.***out***.println("经断言判断，串行计算结果与并发计算结果相同");

}

//判断串行和并发计算结果是否相同

**public** **static** **boolean** MxEqual(**double**[][] MS, **double**[][] MC) {

**for** (**int** i = 0; i<MS.length; i++) {

**for**(**int** j = 0; j<MS[0].length; j++) {

**if** (MS[i][j] != MC[i][j])

**return** **false**;

}

}

**return** **true**;

}

}

**class** MatrixThread **implements** Runnable {

MatrixGenerator mx = **new** MatrixGenerator();

**int** i = mx.i;

**int** j = mx.j;

**int** k = mx.k;

**double**[][] M = **new** **double**[i][k];

**public** **void** Calculate() {

//计算奇数行的矩阵乘法

**if** (Thread.*currentThread*().getName().equals("线程1")) {

**synchronized**(**this**) {

System.***out***.println(Thread.*currentThread*().getName() + " 奇数行 计算结果：");

**for** (**int** x = 0; x<i; x=x+2) {

**for**(**int** y = 0; y<k; y++) {

M[x][y] = 0;

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

M[x][y] += mx.mx1[x][z] \* mx.mx2[z][y];

}

//保留两位小数

BigDecimal bg = **new** BigDecimal(M[x][y]);

M[x][y] = bg.setScale(2, BigDecimal.***ROUND\_HALF\_UP***).doubleValue();

System.***out***.print(M[x][y] + " ");

}

System.***out***.println();

}

System.***out***.println();

}

}

//计算偶数行的矩阵乘法

**if** (Thread.*currentThread*().getName().equals("线程2")) {

**synchronized**(**this**) {

System.***out***.println(Thread.*currentThread*().getName() + " 偶数行 计算结果：");

**for** (**int** x = 1; x<i; x=x+2) {

**for**(**int** y = 0; y<k; y++) {

M[x][y] = 0;

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

M[x][y] += mx.mx1[x][z] \* mx.mx2[z][y];

}

//保留两位小数

BigDecimal bg = **new** BigDecimal(M[x][y]);

M[x][y] = bg.setScale(2, BigDecimal.***ROUND\_HALF\_UP***).doubleValue();

System.***out***.print(M[x][y] + " ");

}

System.***out***.println();

}

System.***out***.println();

}

}

}

@Override

**public** **void** run() {

Calculate();

}

//输出并发计算总结果

**public** **double**[][] print() **throws** InterruptedException {

Thread.*sleep*(3000);

System.***out***.println("并发计算结果：");

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

**for**(**int** n = 0; n<k; n++) {

System.***out***.print(M[m][n]);

System.***out***.print(" ");

}

System.***out***.println();

}

System.***out***.println();

**return** M;

}

**public** **double**[][] ReturnMx1() {

**return** mx.mx1;

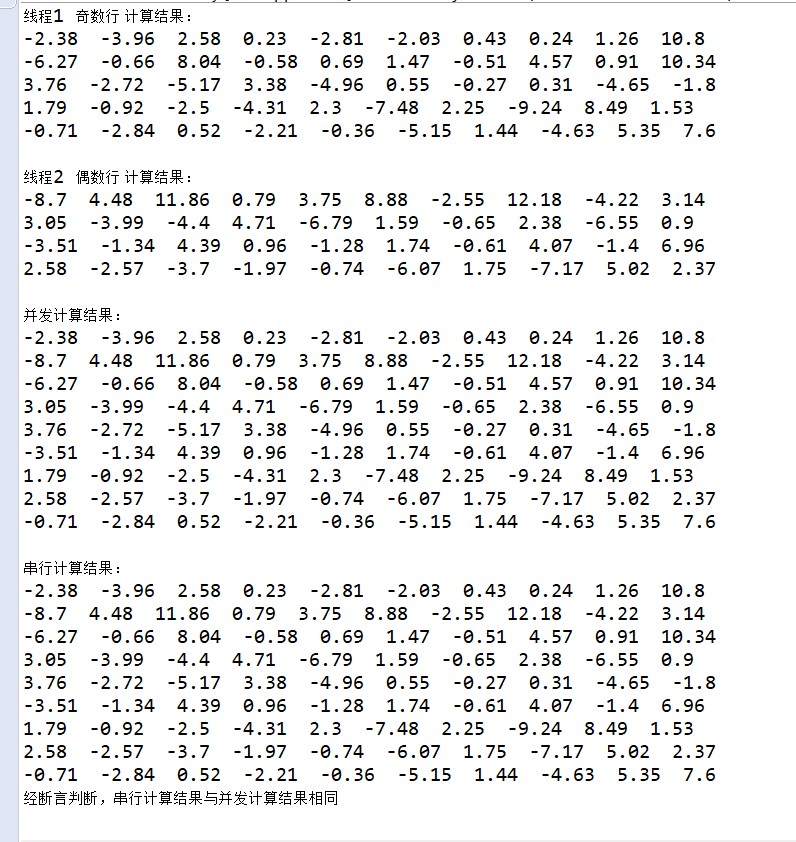
}

**public** **double**[][] ReturnMx2() {

**return** mx.mx2;

}

}

1. **实验结果**