

# 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

## 一、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。

## 二、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 void main(String[] args) {
        MatrixGenerator mx = new MatrixGenerator();
        int i = mx.i;
        int j = mx.j;
        int k = mx.k;
        double[][] M = new double[i][k];

        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] += 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();
        }
    }
}

```

## 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();
        mt.print();
    }
}

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();

    }

    //输出总结果

    public void print() throws InterruptedException {
        Thread.sleep(3000);

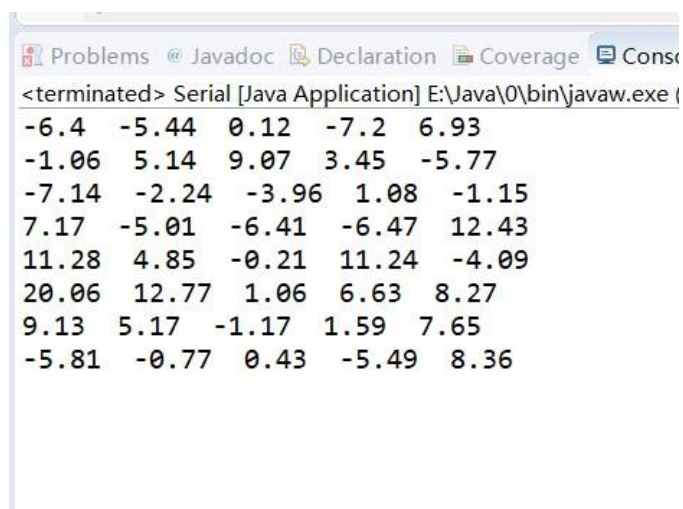
        System.out.println("总计算结果：");

        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();
        }
    }
}

```

### 三、实验结果

串行：



```

<terminated> Serial [Java Application] E:\Java\0\bin\javaw.exe (
-6.4   -5.44  0.12  -7.2   6.93
-1.06  5.14   9.07   3.45  -5.77
-7.14  -2.24  -3.96   1.08  -1.15
7.17   -5.01  -6.41  -6.47  12.43
11.28  4.85   -0.21  11.24  -4.09
20.06  12.77  1.06   6.63   8.27
9.13   5.17  -1.17   1.59   7.65
-5.81  -0.77  0.43   -5.49  8.36

```

并行：

Problems @ Javadoc Declaration Coverage Console

<terminated> Concurrency [Java Application] E:\Java\0\bin\javaw.exe (2020年4月)

线程1 奇数行 计算结果:

-4.14	-7.17	5.25	9.39	-5.16	-4.27	11.74
-6.1	-5.25	-3.47	1.89	-5.19	-5.59	-2.62

线程2 偶数行 计算结果:

4.87	8.37	-0.52	-7.43	6.25	6.99	-7.41
------	------	-------	-------	------	------	-------

总计算结果:

-4.14	-7.17	5.25	9.39	-5.16	-4.27	11.74
4.87	8.37	-0.52	-7.43	6.25	6.99	-7.41
-6.1	-5.25	-3.47	1.89	-5.19	-5.59	-2.62