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。

实验中需要大家分析不同的矩阵大小，不同的线程数，其时间产生的影响，同时也要保证结果的正确性，可以使用串行方法的结果作为标准，与多线程的方法进行比较。

二、设计思想

1、使用MatrixGenerator类生成两个矩阵。

2、使用多线程进行计算。

3、使用串行进行计算。

4、比较两种方法产生的结果

三、具体设计

1、矩阵生成类设计：

```
private int row,column,column2;
private double [] [] array1 = null;
private double [] [] array2 = null;
```

成员变量的声明，包括行列与两个相乘的矩阵。

```
  public MatrixGenerator(int n,int m,int p) {
      this.row = n;
      this.column = m;
      this.column2 = p;
      array1 = new double [row][column];
      array2 = new double [column][column2];
    }
```

使用传递参数的方法生成三个数，做完两个矩阵的行列，因为矩阵相乘要满足第一个的列与第二个的行相等，所以使用同一个数。通过改变传递的

```
    public void getMatrix() {
      for(int i=0;i<this.row;i++) {
        for(int j=0;j<this.column;j++) {
          array1[i][j]=Math.abs(ran.nextDouble()*1000);
        }
      }
      for(int i=0;i<this.column;i++) {
        for(int j=0;j<this.column2;j++) {
          array2[i][j]=Math.abs(ran.nextDouble()*1000);
        }
      }
    }
```

给两个矩阵赋值，使用nextDouble()方法，生成0-1000的浮点数。

2、计算类设计

(1)串行计算方法实现

```
 //串行方法
      for(int i=0;i<m;i++){
          for(int j=0;j<p;j++){
               for(int k=0;k<n;k++){
              c[i][j] =c[i][j] + A[i][k] * B[k][j];
             }
          }
      }
```

直接使用三重循环进行计算。

(2)并行计算方法实现

```
//子线程数量
int threadNum = 10;
//子线程的分片计算间隔
int gap = A.length / threadNum;
CountDownLatch countDownLatch = new CountDownLatch(threadNum);
```

先对子线程个数进行定义，开始定义为10个子线程，之后根据矩阵的大小对矩阵进行分块，即定义一个线程负责的矩阵大小。

通过CountDownLatch对线程进行定义。

```
public CalculateTask(double[][] A, double[][] B, int index, int gap, double[]
[] result, CountDownLatch countDownLatch) {
    this.A = A;
    this.B = B;
    this.index = index;
    this.gap = gap;
    this.result = result;
    this.countDownLatch = countDownLatch;
  }
```

线程的声明，定义了定义了两个计算矩阵，计算起始位置，间隔，并保存结果。

```
public void run() {
    // TODO Auto-generated method stub
    for (int i = index * gap; i < (index + 1) * gap; i++)
      for (int j = 0; j < B[0].length; j++) {
        for (int k = 0; k < B.length; k++)
          result[i][j] += A[i][k] * B[k][j];
      }
      // 线程数减1
    countDownLatch.countDown();
  }
```

run方法的定义，在每个gap里分别对A,B矩阵相乘，每计算完一个区间将线程数减1。

```
    boolean flag = true;
    for(int i=0;i<A.length;i++) {
        for(int j=0;j<B[0].length;j++) {
            if(parallel_result[i][j]!=serial_result[i][j]) {
              flag = false;
            }
        }
     }
    if(flag) {
        System.out.println("结果一致");
    }
```

判断两种方法生成的矩阵是否相等。

四、实现结果

(1) 开始设定两个矩阵分别为300 * 300 300 * 300，子线程数为5

结果一致

看两个的时间：

```
请输入两个矩阵的行与列
300 300 300
串行计算开始时刻:1588148939806
串行计算结束时刻:1588148939868
串行计算运行时间:62
并行计算开始时刻:1588148939868
并行计算结束时刻:1588148939895
并行计算运行时间:27
```

(2)改变矩阵大小为 300 * 500 500 * 400

(3)再次改变矩阵大小为 800 * 800 800 * 800

```
请输入两个矩阵的行与列
800 800 800
串行计算开始时刻:1588149042696
串行计算结束时刻:1588149043573
串行计算运行时间:877
并行计算开始时刻:1588149043573
并行计算结束时刻:1588149043787
并行计算运行时间:214
```

(4)再次改变矩阵大小为 1000 * 1000 1000 * 1000

```
<terminated> Calu [Java Application] /Library/Java/JavaVirtualMachines/jdk-12.0.2.jdk/Contents
请输入两个矩阵的行与列
1000 1000 1000
串行计算开始时刻:1588149469625
串行计算结束时刻:1588149471558
串行计算运行时间:1933
并行计算开始时刻:1588149471558
并行计算结束时刻:1588149471961
并行计算运行时间:403
结果一致
```

结果得出：随着矩阵的增大，多线程计算的时间更短

修改线程数为10

(5)设定两个矩阵分别为300 * 300 300 * 300

```
<terminated> Calu [Java Application] /Library/Java/JavaVirtualMachines/jdk-12.0.2.jdk/Contents/Hom
请输入两个矩阵的行与列
300 300 300
串行计算开始时刻:1588149209993
串行计算结束时刻:1588149210053
串行计算运行时间:60
并行计算开始时刻:1588149210054
并行计算结束时刻:1588149210087
并行计算运行时间:33
结果一致
```

发现反而比5个线程更慢了

(6)改变矩阵大小为 300 * 500 500 * 400

```
请输入两个矩阵的行与列
300 500 400
串行计算开始时刻:1588149341273
串行计算结束时刻:1588149341395
串行计算运行时间:122
并行计算开始时刻:1588149341395
并行计算结束时刻:1588149341455
并行计算运行时间:60
结果一致
```

(7)再次改变矩阵大小为 800 * 800 800 * 800

```
请输入两个矩阵的行与列
800 800 800
串行计算开始时刻:1588149398309
串行计算结束时刻:1588149399176
串行计算运行时间:867
并行计算开始时刻:1588149399176
并行计算结束时刻:1588149399390
并行计算运行时间:214
结果一致
```

(8)再次改变矩阵大小为 1000 * 1000 1000 * 1000

```
请输入两个矩阵的行与列
1000 1000 1000
串行计算开始时刻:1588149520844
串行计算结束时刻:1588149522949
串行计算运行时间:2105
并行计算开始时刻:1588149522950
并行计算结束时刻:1588149523599
并行计算运行时间:649
结果一致
```

结论：随着矩阵大小的变化，多线程效率明显变高。但当线程数增多时，矩阵过小时运算时间变化不明显，矩阵变大时效率更高。但每次此时矩阵元素有随机性。