**JAVA编程进阶上机报告**

****

第四次上机作业

**学 院\_\_\_\_\_计算机学院\_\_\_\_\_\_\_\_**

**专 业\_\_\_\_\_\_\_计算机科学\_\_\_\_\_\_\_**

**姓 名\_\_\_\_\_\_罗多福\_\_\_\_\_\_\_\_**

**学 号\_\_\_\_\_\_\_6317000036\_\_\_\_\_\_\_**

**年 级\_\_\_\_\_\_\_\_2017\_\_\_\_\_\_**

**班 级\_\_\_\_\_\_\_\_计算机五班\_\_\_\_\_\_**

# 实验要求

## 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

**package** lab\_4;

**import** java.math.\*;

**public** **class** MatrixMultiplication {

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

**int** m;

**int** n;

Double[][] Matrix;

**public** Double[][] getMatrix(**int** m, **int** n){

**this**.m = m;

**this**.n = n;

Matrix = **new** Double[m][];

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

Matrix[i] = **new** Double[n];

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

Matrix[i][j] = Math.*random*()\*10000;

}

}

**return** Matrix;

}

**public** **void** display(Double[][] a, **int** m, **int** n) {

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

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

System.***out***.print(a[i][j] + " ");

}

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

}

}

}

**static** **class** MultiplyPart **implements** Runnable{

Double[][] a,b,c;

**int** i, o, n;

MultiplyPart(Double[][] a, Double[][] b, Double[][] c, **int** i, **int** o, **int** n){

**this**.a = a;

**this**.b = b;

**this**.c = c;

**this**.i = i;

**this**.o = o;

**this**.n = n;

}

@Override

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

c[i] = **new** Double[o];

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

c[i][j] = 0.0;

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

c[i][j] += a[i][k]\*b[k][j];

}

}

}

}

**static** **public** Double[][] serialMatrixMultiply(Double[][] a, Double[][] b,**int** m, **int** n, **int** o) {

Double[][] result = **new** Double[m][];

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

result[i] = **new** Double[o];

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

result[i][j] = 0.0;

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

result[i][j] += a[i][k]\*b[k][j];

}

}

}

**return** result;

}

**static** **public** Double[][] parallelMatrixMultiply(Double[][] a, Double[][] b,**int** m, **int** n, **int** o) {

Double[][] result = **new** Double[m][];

Thread[] threads = **new** Thread[m];

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

Runnable s = **new** MultiplyPart(a, b, result, i, o, n);

Thread t = **new** Thread(s);

t.start();

threads[i] = t;

}

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

**try** {

threads[i].join();

}**catch**(Exception e) {

e.printStackTrace();

}

}

**return** result;

}

**static** **public** **void** main(String[] avgs) {

MatrixGenerator matrixGenerator = **new** MatrixGenerator();

**int** m=800,n=1000,o=600;

Double[][] a = matrixGenerator.getMatrix(m, n);

Double[][] b = matrixGenerator.getMatrix(n, o);

Long startTime;

startTime=System.*currentTimeMillis*();

Double[][] result1 = *serialMatrixMultiply*(a,b,m,n,o);

System.***out***.println("串行计算所用时间:" + (System.*currentTimeMillis*()-startTime)+"ms");

startTime = System.*currentTimeMillis*();

Double[][] result2 = *parallelMatrixMultiply*(a,b,m,n,o);

System.***out***.println("并行计算所用时间:" + (System.*currentTimeMillis*()-startTime)+"ms");

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

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

**if**(Math.*abs*(result1[i][j]-result2[i][j])>0.00001) {

System.***out***.println("计算错误");

**break**;

}

}

}

}

}

1. 运行结果
2. 

结果分析：

并行运行时间要远小于串行运行，因为串行的程序段必须按照先后顺序来执行，也就是只有前面的程序段执行完了，后面的程序段才能执行，浪费了CPU资源；并行执行中，JVM可以把时间分配给不同的程序段，提高运行效率。