# 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 3 班**

**学 号 3018216157**

**姓 名 郑开**

1. **实验要求**
2. 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。
3. 使用串行方式实现矩阵乘法。
4. 使用多线程方式实现矩阵乘法。
5. 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。
6. **源代码**

* MatrixGenerator.java

package matrix;   
   
import java.util.Random;   
   
public class MatrixGenerator {   
 public static Matrix generate(int rowNum, int columnNum) {   
 Random random = new Random(System.currentTimeMillis());   
 Matrix matrix = new Matrix(rowNum,columnNum);   
 for (int i = 0; i < rowNum; i++) {   
 for (int j = 0; j < columnNum; j++) {   
 matrix.setValue(i,j,random.nextInt(100));   
 }   
 }   
 return matrix;   
 }   
   
}

* Matrix.java

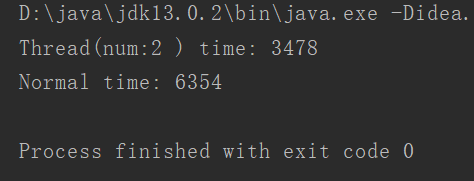
package matrix;   
   
import java.util.concurrent.CountDownLatch;   
   
public class Matrix {   
 private double[][] doubles;   
 private int row;   
 private int column;   
   
 public Matrix(int row, int column) {   
 this.row = row;   
 this.column = column;   
 doubles = new double[row][column];   
 }   
   
 public void setValue(int row, int column, double value) {   
 doubles[row][column] = value;   
 }   
   
 public void print() {   
 for (int i = 0; i < row; i++) {   
 for (int j = 0; j < column; j++) {   
 System.out.print(doubles[i][j] + " ");   
 }   
 System.out.println();   
 }   
 }   
   
 public double getValue(int row, int column) {   
 return doubles[row][column];   
 }   
   
 public Matrix multiply(Matrix matrix) throws Exception {   
 if (this.column != matrix.row) {   
 throw new Exception("矩阵乘法无效 需满足a\*b b\*c的格式");   
 }   
 Matrix result = new Matrix(this.row, matrix.column);   
 for (int i = 0; i < row; i++) {   
 for (int j = 0; j < matrix.column; j++) {   
 result.setValue(i, j, getXy(i, j, matrix));   
 }   
 }   
 return result;   
 }   
   
 public Matrix multiply(Matrix matrix, int threadNum) throws Exception {   
 if (threadNum < 0) {   
 throw new Exception("param threadNum < 0");   
 }   
 if (this.column != matrix.row) {   
 throw new Exception("矩阵乘法无效 需满足a\*b b\*c的格式");   
 }   
   
 final CountDownLatch countDownLatch = new CountDownLatch(threadNum);   
 Matrix result = new Matrix(this.row, matrix.column);   
 for (int i = 0; i < threadNum; i++) {   
 final int r = i;   
 Thread thread = new Thread(() -> {   
 for (int j = r; j < result.row; j += threadNum) {   
 for (int k = 0; k < result.column; k++) {   
 result.setValue(j, k, getXy(j, k, matrix));   
 }   
 }   
 countDownLatch.countDown();   
 });   
 thread.start();   
 }   
 countDownLatch.await();   
 return result;   
 }   
   
 private double getXy(int x, int y, Matrix matrix) {   
 double result = 0;   
 int i = 0;   
 int j = 0;   
 while (i < column && j < matrix.row) {   
 result += this.getValue(x, i) \* matrix.getValue(j, y);   
 i++;   
 j++;   
 }   
 return result;   
 }   
}

* Main.java

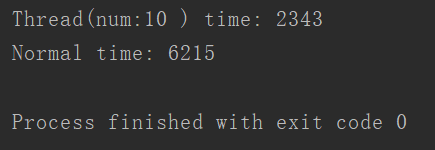
import matrix.Matrix;   
import matrix.MatrixGenerator;   
   
public class Main {   
 private static final int THREAD\_NUM = 2;   
   
 public static void main(String[] args) throws Exception {   
 Matrix matrix1 = MatrixGenerator.generate(1000, 1000);   
 Matrix matrix2 = MatrixGenerator.generate(1000, 1000);   
   
 long threadStart = System.currentTimeMillis();   
 Matrix matrix3 = matrix1.multiply(matrix2, THREAD\_NUM);   
 long threadEnd = System.currentTimeMillis();   
   
 long normalStart = System.currentTimeMillis();   
 Matrix matrix4 = matrix1.multiply(matrix2);   
 long normalEnd = System.currentTimeMillis();   
   
 System.out.println("Thread(num:" + THREAD\_NUM + " ) time: " + (threadEnd - threadStart));   
 System.out.println("Normal time: " + (normalEnd - normalStart));   
   
 }   
}

1. **实验结果**

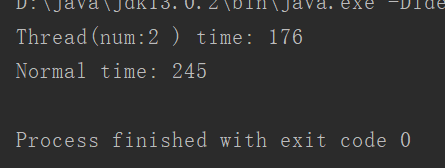
当矩阵大小为1000\*1000时，开启2个线程和单独一个线程的结果：



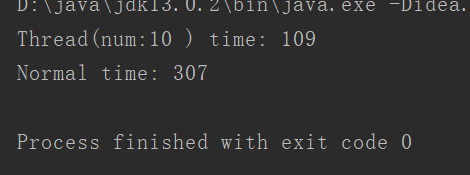
当矩阵大小为1000\*1000时，开启10个线程和单独一个线程的结果：



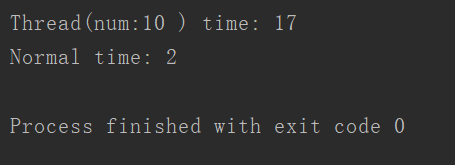
当矩阵大小为500\*500时，开启2个线程和单独一个线程的结果：



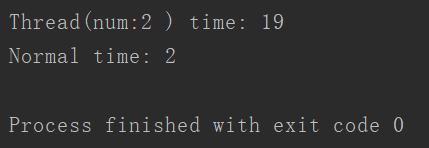
当矩阵大小为500\*500时，开启10个线程和单独一个线程的结果：



当矩阵大小为100\*100时，开启2个线程和单独一个线程的结果：



当矩阵大小为100\*100时，开启10个线程和单独一个线程的结果：



实验结果可得，当矩阵较大时，多个线程比单线程运算速度快，当矩阵较小时，单线程比多线程运算速度快。

产生该现象的原因：创建线程需要消耗计算机资源，该过程需要一定时间。而且，当矩阵较小时，计算量也小，多线程所节省的运算时间比线程调度的时间少，多线程计算时间则会慢。而当矩阵较大时，计算量较大，多线程运行将充分调度处理器资源，发挥多核的优势，提高计算速度。