# JAVA编程进阶上机报告

****

**学 院 智能与计算学部**

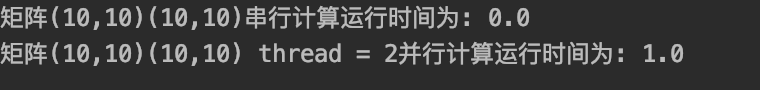
**专 业 软件工程**

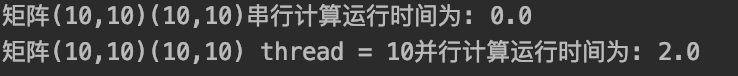
**班 级 5班**

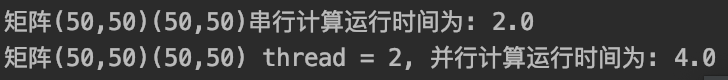
**学 号 3018216259**

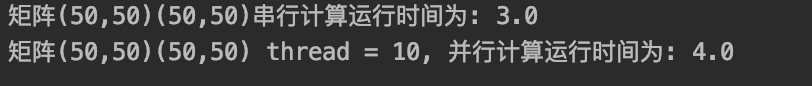
**姓 名 李澄**

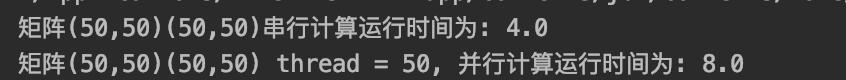
1. **实验要求**
2. 编写矩阵随机生成类 MatrixGenerator 类，随机生成任意大小的矩阵，矩阵单元使用 double 存储。
3. 使用串行方式实现矩阵乘法。
4. 使用多线程方式实现矩阵乘法。
5. 比较串行和并行两种方式使用的时间，利用第三次使用中使用过的 jvm状态查看命令，分析产生时间差异的原因是什么。
6. **源代码**
7. public class MatrixGenerator {  
    //生成一个n阶矩阵A,并打印A  
    public static double[][] create\_Matrix(int row\_number,int column\_number ) {  
     
     
    double matrixA[][] = new double[row\_number][column\_number];  
    for (int i = 0; i <= row\_number - 1; i++) {  
    for (int j = 0; j <= column\_number - 1; j++) {  
    matrixA[i][j] = *random\_number*();  
    System.*out*.print(matrixA[i][j] + " ");  
    }  
    System.*out*.println();  
    }  
    return matrixA;  
    }  
     
    //产生一个1~100的随机整数，用于自动填入矩阵A的每个位之中  
    public static int random\_number(){  
    int number;  
    number = new java.util.Random().nextInt(100)+1;  
    return number;  
    }  
     
   }
8. public class MatrixMultiply extends Thread {  
    private double[][] matrixA,matrixB;  
    private int gap;  
    private int index;  
    private CountDownLatch countDownLatch;  
    private double[][] result;  
     
    public static double[][] serialMultipply(double[][] matrixA, double[][] matrixB) {  
    double startTime = System.*currentTimeMillis*();  
    if(matrixA[0].length != matrixB.length){  
    System.*out*.println("两个矩阵不能相乘");  
    return null;  
    }  
    int x = matrixA.length;  
    int y = matrixB[0].length;  
    double[][] result = new double[x][y];  
    for (int i = 0; i < x; i++) {  
    for (int j = 0; j < y; j++) {  
    result[i][j] = 0;  
    for (int k = 0; k < matrixA[0].length; k++) {  
    result[i][j] += matrixA[i][k] \* matrixB[k][j];  
    }  
    }  
    }  
    double endTime = System.*currentTimeMillis*();  
    System.*out*.println("串行计算运行时间为: " + (endTime - startTime));  
    return result;  
    }  
     
    public MatrixMultiply(double[][] A, double[][] B, int index, int gap, double[][] result, CountDownLatch countDownLatch) {  
    this.matrixA = A;  
    this.matrixB = B;  
    this.index = index;  
    this.gap = gap;  
    this.result = result;  
    this.countDownLatch = countDownLatch;  
    }  
     
    @Override  
    public void run() {  
    for (int i = index \* gap; i < (index + 1) \* gap; i++) {  
    for (int j = 0; j < matrixB[0].length; j++) {  
    for (int k = 0; k < matrixB.length; k++) {  
    result[i][j] += matrixA[i][k] \* matrixB[k][j];  
    }  
    }  
    }  
    countDownLatch.countDown();  
    }  
     
    public static double[][] parallelMultiply(double[][] matrixA, double[][] matrixB, int threadnum) throws InterruptedException {  
    double startTime = System.*currentTimeMillis*();  
    int x = matrixA.length;  
    int y = matrixB[0].length;  
    double[][] result = new double[x][y];  
    int gap = matrixA.length / threadnum;  
    CountDownLatch countDownLatch = new CountDownLatch(threadnum);  
    for (int i = 0; i < threadnum; i++) {  
    MatrixMultiply task = new MatrixMultiply(matrixA,matrixB,i,gap,result,countDownLatch);  
    task.start();  
    }  
    countDownLatch.await();  
    double endTime = System.*currentTimeMillis*();  
    System.*out*.println("并行计算运行时间为: " + (endTime - startTime));  
    return result;  
    }  
     
    public static void main(String[] args) throws InterruptedException {  
    double[][] matrix1 = MatrixGenerator.*create\_Matrix*(10,10);  
    double[][] martrix2 = MatrixGenerator.*create\_Matrix*(10,10);  
     
    double[][] serialRes = *serialMultipply*(matrix1, martrix2);  
    double[][] parallelRes = *parallelMultiply*(matrix1,martrix2,2);  
    assert Arrays.*deepEquals*(serialRes, parallelRes);  
    }  
     
   }
9. **实验结果**

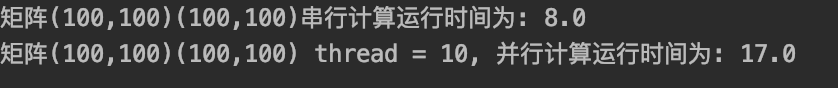
****

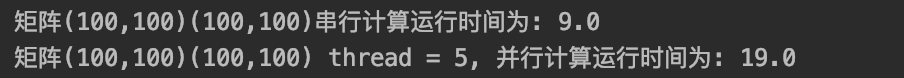
****

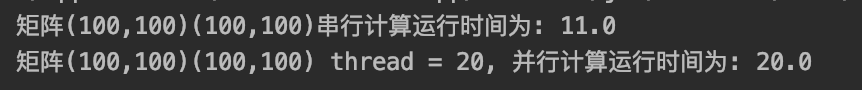
****

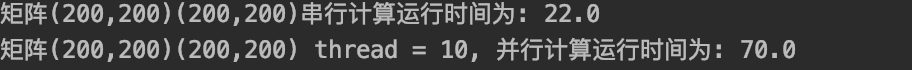
****

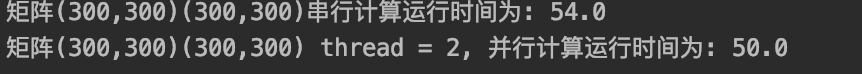
****

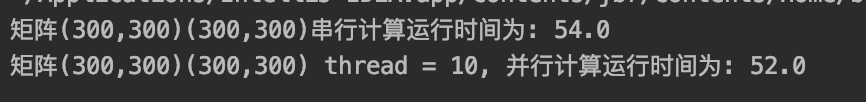
****

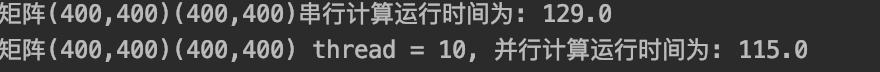
****

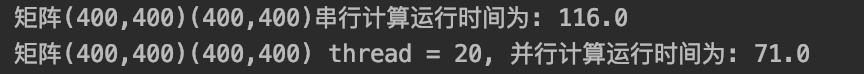
****

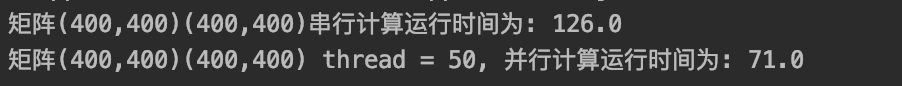
****

****

****

****

****

****

**由以上实验结果可知，当矩阵较小时，串行计算要比并行计算快，因为创建线程也需要时间；而随着矩阵逐渐增大，在300阶矩阵的乘法计算中，并行计算要快于串行，随着线程数的增加，创建线程的开销也加大，逐渐并行计算的时间要加长。总而言之，在矩阵较小时，并行计算不存在时间上的优势，随着矩阵的增大，并行计算优势凸显出来，但线程数的增加也会加大线程的开销，所以线程数在到达某一个峰值时，若再增加线程数，时间就会变长。**