# 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 3018216232**

**姓 名 刘 昕**

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。

源代码

**package** java\_advance.lab4;

**public** **class** Matrix {

**private** **double** [][] matrix;

**private** **int** row, col;

**public** Matrix(**int** row, **int** col)

{

**this**.row = row;

**this**.col = col;

**this**.matrix = **new** **double**[row][col];

}

**public** **double** getMatrix(**int** row, **int** col)

{

**return** matrix[row][col];

}

**public** **int** getRow()

{

**return** row;

}

**public** **int** getCol()

{

**return** col;

}

**public** **void** setMatrix(**int** row, **int** col, **double** a)

{

**if** (row <= **this**.row && col <= **this**.col)

{

**this**.matrix[row][col] = a;

}

}

}

**package** java\_advance.lab4;

**import** java.util.Random;

**public** **class** MatrixGenerator

{

**private** Matrix matrix;

**public** MatrixGenerator()

{

Random temp = **new** Random();

**int** row = temp.nextInt(100);

**int** col = temp.nextInt(100);

**this**.matrix = **new** Matrix(row, col);

**this**.initMatrix();

}

**public** MatrixGenerator(**int** row, **int** col)

{

**this**.matrix = **new** Matrix(row, col);

**this**.initMatrix();

}

**public** **void** initMatrix()

{

Random temp = **new** Random();

**for** (**int** i = 0; i < **this**.matrix.getRow(); i++)

{

**for** (**int** j = 0; j < **this**.matrix.getCol(); j++)

{

**this**.matrix.setMatrix(i, j, temp.nextInt(100));

}

}

}

**public** Matrix getMatrix()

{

**return** **this**.matrix;

}

}

**package** java\_advance.lab4;

**class** TwoThread **implements** Runnable

{

Matrix x, y, result;

**public** TwoThread(Matrix x, Matrix y, Matrix result)

{

**this**.x = x;

**this**.y = y;

**this**.result = result;

}

@Override

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

{

**if** (Thread.*currentThread*().getName().equals("线程1"))

{

firstThread();

}

**else** **if** (Thread.*currentThread*().getName().equals("线程2"))

{

secondThread();

}

}

**public** **void** firstThread()

{

**for** (**int** i = 0; i < x.getRow(); i += 2)

{

**for** (**int** j = 0; j < y.getCol(); j++)

{

**double** sum = 0;

**for** (**int** k = 0; k < x.getCol(); k++)

{

sum += x.getMatrix(i, k) \* y.getMatrix(k, j);

}

result.setMatrix(i, j, sum);

}

}

}

**public** **void** secondThread()

{

**for** (**int** i = 1; i < x.getRow(); i += 2)

{

**for** (**int** j = 0; j < y.getCol(); j++)

{

**double** sum = 0;

**for** (**int** k = 0; k < x.getCol(); k++)

{

sum += x.getMatrix(i, k) \* y.getMatrix(k, j);

}

result.setMatrix(i, j, sum);

}

}

}

}

**package** java\_advance.lab4;

**class** FourThread **implements** Runnable

{

Matrix x, y, result;

**public** FourThread(Matrix x, Matrix y, Matrix result)

{

**this**.x = x;

**this**.y = y;

**this**.result = result;

}

@Override

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

{

**if** (Thread.*currentThread*().getName().equals("线程1"))

{

firstThread();

}

**else** **if** (Thread.*currentThread*().getName().equals("线程2"))

{

secondThread();

}

**else** **if** (Thread.*currentThread*().getName().equals("线程3"))

{

thirdThread();

}

**else** **if** (Thread.*currentThread*().getName().equals("线程4"))

{

fourthThread();

}

}

**public** **void** firstThread()

{

**for** (**int** i = 0; i < x.getRow(); i += 4)

{

**for** (**int** j = 0; j < y.getCol(); j++)

{

**double** sum = 0;

**for** (**int** k = 0; k < x.getCol(); k++)

{

sum += x.getMatrix(i, k) \* y.getMatrix(k, j);

}

result.setMatrix(i, j, sum);

}

}

}

**public** **void** secondThread()

{

**for** (**int** i = 1; i < x.getRow(); i += 4)

{

**for** (**int** j = 0; j < y.getCol(); j++)

{

**double** sum = 0;

**for** (**int** k = 0; k < x.getCol(); k++)

{

sum += x.getMatrix(i, k) \* y.getMatrix(k, j);

}

result.setMatrix(i, j, sum);

}

}

}

**public** **void** thirdThread()

{

**for** (**int** i = 2; i < x.getRow(); i += 4)

{

**for** (**int** j = 0; j < y.getCol(); j++)

{

**double** sum = 0;

**for** (**int** k = 0; k < x.getCol(); k++)

{

sum += x.getMatrix(i, k) \* y.getMatrix(k, j);

}

result.setMatrix(i, j, sum);

}

}

}

**public** **void** fourthThread()

{

**for** (**int** i = 3; i < x.getRow(); i += 4)

{

**for** (**int** j = 0; j < y.getCol(); j++)

{

**double** sum = 0;

**for** (**int** k = 0; k < x.getCol(); k++)

{

sum += x.getMatrix(i, k) \* y.getMatrix(k, j);

}

result.setMatrix(i, j, sum);

}

}

}

}

**package** java\_advance.lab4;

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

**static** Matrix *x*;

**static** Matrix *y*;

**public** MatrixMultiplication(Matrix x, Matrix y)

{

**this**.*x* = x;

**this**.*y* = y;

}

**public** **static** Matrix multiplyBySequentially(Matrix x, Matrix y)

{

**int** xrow = x.getRow();//a

**int** xcol = x.getCol();//b1

**int** yrow = y.getRow();//b2

**int** ycol = y.getCol();//c

**if** (xcol == yrow)

{

Matrix result = **new** Matrix(xrow, ycol);

**for** (**int** i = 0; i < xrow; i++)

{

**for** (**int** j = 0; j < ycol; j++)

{

**double** sum = 0;

**for** (**int** k = 0; k < xcol; k++)

{

sum += x.getMatrix(i, k) \* y.getMatrix(k, j);

}

result.setMatrix(i, j, sum);

}

}

**return** result;

}

**else**

{

**return** **null**;

}

}

**static** Matrix multiplyByTwoThread(Matrix x, Matrix y) **throws** InterruptedException

{

**int** xrow = x.getRow();

**int** xcol = x.getCol();

**int** yrow = y.getRow();

**int** ycol = y.getCol();

**if** (xcol == yrow)

{

Matrix result = **new** Matrix(xrow, ycol);

TwoThread tt = **new** TwoThread(x, y, result);

Thread thread1 = **new** Thread(tt, "线程1");

Thread thread2 = **new** Thread(tt, "线程2");

thread1.start();

thread2.start();

**while** (thread1.isAlive() || thread2.isAlive()){}

**return** result;

}

**else**

{

**return** **null**;

}

}

**static** Matrix multiplyByFourThread(Matrix x, Matrix y) **throws** InterruptedException

{

**int** xrow = x.getRow();

**int** xcol = x.getCol();

**int** yrow = y.getRow();

**int** ycol = y.getCol();

**if** (xcol == yrow)

{

Matrix result = **new** Matrix(xrow, ycol);

FourThread tt = **new** FourThread(x, y, result);

Thread thread1 = **new** Thread(tt, "线程1");

Thread thread2 = **new** Thread(tt, "线程2");

Thread thread3 = **new** Thread(tt, "线程3");

Thread thread4 = **new** Thread(tt, "线程4");

thread1.start();

thread2.start();

thread3.start();

thread4.start();

**while** (thread1.isAlive() || thread2.isAlive() || thread3.isAlive() || thread4.isAlive()){}

**return** result;

}

**else**

{

**return** **null**;

}

}

}

**package** java\_advance.lab4;

**public** **class** test

{

**public** **static** **void** main(String[] args) **throws** InterruptedException

{

**int** size = 100;

Matrix x = **new** MatrixGenerator(size, size).getMatrix();

Matrix y = **new** MatrixGenerator(size, size).getMatrix();

**long** time1 = System.*nanoTime*();

Matrix resultSequentially = MatrixMultiplication.*multiplyBySequentially*(x, y);

**long** time2 = System.*nanoTime*();

Matrix resultParallelTwoThread = MatrixMultiplication.*multiplyByTwoThread*(x, y);

**long** time3 = System.*nanoTime*();

Matrix resultParallelFourThread = MatrixMultiplication.*multiplyByFourThread*(x, y);

**long** time4 = System.*nanoTime*();

**assert** resultSequentially.equals(resultParallelTwoThread);

**assert** resultSequentially.equals(resultParallelFourThread);

System.***out***.println("当矩阵大小: " + size + " \* " + size);

System.***out***.println("串行用时: " + (time2 - time1) + "ns");

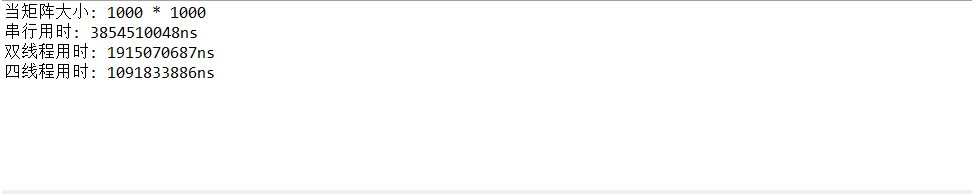
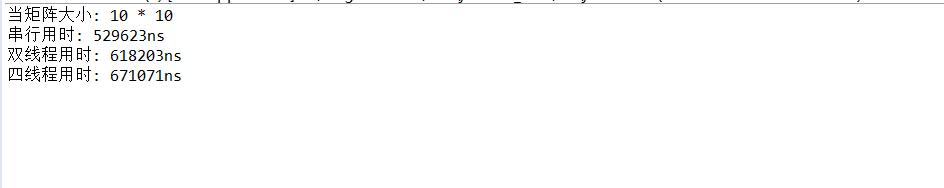
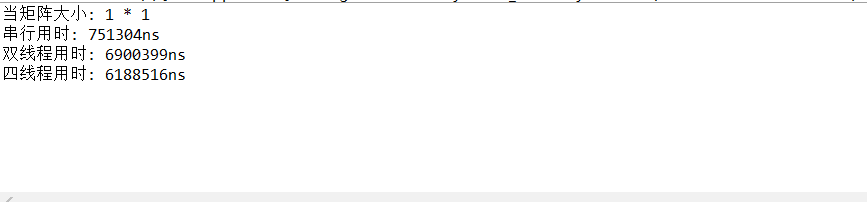
System.***out***.println("双线程用时: " + (time3 - time2) + "ns");

System.***out***.println("四线程用时: " + (time4 - time3) + "ns");

}

}

运行结果



可以看到，当矩阵较大时，多线程计算比串行计算要快，多线程并发会调用更多资源，效率更高

当矩阵较小时，串行计算比多线程计算要快，因为计算的资源时间开销不大，而多线程需要花费额外的资源，已完成线程创建调度等。