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　五班

学　号　3018216242

姓　名　邢思洋

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。

二、源代码

Matrix.java:

```
package xsy.lab4;

import java.util.Arrays;

public class Matrix
{
    private double [][] matrix;
    private int m, n;

    public Matrix(int m, int n)
    {
        this.m = m;
        this.n = n;
        this.matrix = new double[m][n];
    }

    public double[][] getMatrix()
    {
        return matrix;
    }

    public double getMatrix(int m, int n)
    {
        return matrix[m][n];
    }

    public void setMatrix(int i, int j, double a)
    {
        if (i <= this.m && j <= this.n)
        {
```

```
            this.matrix[i][j] = a;
        }
    }

    public void printMatrix()
    {
        for (int i = 0; i < this.m; i++)
        {
            for (int j = 0; j < this.n; j++)
            {
                System.out.print(this.matrix[i][j] + " ");
            }
            System.out.println();
        }
    }

    public int getM()
    {
        return m;
    }

    public int getN()
    {
        return n;
    }

    @Override
    public boolean equals(Object obj)
    {
        if (obj == null)
        {
            return false ;
        }
        else
        {
            if (obj instanceof Matrix)
            {
                Matrix c = (Matrix) obj;
                if (this.m != c.getM() || this.n != c.getN())
                {
                    return false;
                }
                else
                {
```

```
                for (int i = 0; i < this.m; i++)
                {
                    for (int j = 0; j < this.n; j++)
                    {
                        if (this.matrix[i][j] != c.getMatrix(i, j))
                        {
                            return false;
                        }
                    }
                }
                return true;
            }
        }
        else
        {
            return false;
        }
    }
}
```

MatrixGenerator.java:

```
package xsy.lab4;

import java.util.Random;

public class MatrixGenerator
{
    private Matrix matrix;

    public MatrixGenerator(int m, int n)
    {
        this.matrix = new Matrix(m, n);
        this.initMatrix();
    }

    public void initMatrix()
    {
        Random r = new Random();
        for (int i = 0; i < this.matrix.getM(); i++)
        {
            for (int j = 0; j < this.matrix.getN(); j++)
            {
```

```
                this.matrix.setMatrix(i, j, r.nextInt(100));
            }
        }
    }

    public Matrix getMatrix()
    {
        return this.matrix;
    }
}
```

MatrixMultiplication.java:

```
package xsy.lab4;

public class MatrixMultiplication
{
    public static Matrix multiplySequentially(Matrix x, Matrix y)
    {
        int a = x.getM();
        int b1 = x.getN();
        int b2 = y.getM();
        int c = y.getN();
        if (b1 == b2)
        {
            Matrix result = new Matrix(a, c);
            for (int i = 0; i < a; i++)
            {
                for (int j = 0; j < c; j++)
                {
                    double sum = 0;
                    for (int k = 0; k < b1; k++)
                    {
                        sum += x.getMatrix(i, k) * y.getMatrix(k, j);
                    }
                    result.setMatrix(i, j, sum);
                }
            }
            return result;
        }
        else
        {
            return null;
        }
```

```
    }

    public static Matrix multiplyParallelTwoThread(Matrix x, Matrix
y) throws InterruptedException
    {
        int a = x.getM();
        int b1 = x.getN();
        int b2 = y.getM();
        int c = y.getN();
        if (b1 == b2)
        {
            Matrix result = new Matrix(a, c);
            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();
//          thread1.join();
            thread2.start();
//          thread2.join();
            while (thread1.isAlive() || thread2.isAlive()){}
            return result;
        }
        else
        {
            return null;
        }
    }

    public static Matrix multiplyParallelThreeThread(Matrix x, Matrix
y) throws InterruptedException
    {
        int a = x.getM();
        int b1 = x.getN();
        int b2 = y.getM();
        int c = y.getN();
        if (b1 == b2)
        {
            Matrix result = new Matrix(a, c);
            ThreeThread tt = new ThreeThread(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");
            thread1.start();
//          thread1.join();
```

```
            thread2.start();
//          thread2.join();
            thread3.start();
//          thread3.join();
            while (thread1.isAlive() || thread2.isAlive() ||
thread3.isAlive()){}
            return result;
        }
        else
        {
            return null;
        }
    }

    public static Matrix multiplyParallelFourThread(Matrix x, Matrix
y) throws InterruptedException
    {
        int a = x.getM();
        int b1 = x.getN();
        int b2 = y.getM();
        int c = y.getN();
        if (b1 == b2)
        {
            Matrix result = new Matrix(a, c);
            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();
//          thread1.join();
            thread2.start();
//          thread2.join();
            thread3.start();
//          thread3.join();
            thread4.start();
//          thread4.join();
            while (thread1.isAlive() || thread2.isAlive() ||
thread3.isAlive() || thread4.isAlive()){}
            return result;
        }
        else
        {
            return null;
```

```
        }
    }
}

class TwoThread implements Runnable
{
    Matrix matrix1, matrix2, result;

    public TwoThread(Matrix matrix1, Matrix matrix2, Matrix result)
    {
        this.matrix1 = matrix1;
        this.matrix2 = matrix2;
        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 < matrix1.getM(); i += 2)
        {
            for (int j = 0; j < matrix2.getN(); j++)
            {
                double sum = 0;
                for (int k = 0; k < matrix1.getN(); k++)
                {
                    sum += matrix1.getMatrix(i, k) *
matrix2.getMatrix(k, j);
                }
                result.setMatrix(i, j, sum);
            }
        }
    }
```

```
    public void secondThread()
    {
        for (int i = 1; i < matrix1.getM(); i += 2)
        {
            for (int j = 0; j < matrix2.getN(); j++)
            {
                double sum = 0;
                for (int k = 0; k < matrix1.getN(); k++)
                {
                    sum += matrix1.getMatrix(i, k) *
matrix2.getMatrix(k, j);
                }
                result.setMatrix(i, j, sum);
            }
        }
    }
}

class ThreeThread implements Runnable
{
    Matrix matrix1, matrix2, result;

    public ThreeThread(Matrix matrix1, Matrix matrix2, Matrix result)
    {
        this.matrix1 = matrix1;
        this.matrix2 = matrix2;
        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();
```

```
        }
    }

    public void firstThread()
    {
        for (int i = 0; i < matrix1.getM(); i += 3)
        {
            for (int j = 0; j < matrix2.getN(); j++)
            {
                double sum = 0;
                for (int k = 0; k < matrix1.getN(); k++)
                {
                    sum += matrix1.getMatrix(i, k) *
matrix2.getMatrix(k, j);
                }
                result.setMatrix(i, j, sum);
            }
        }
    }

    public void secondThread()
    {
        for (int i = 1; i < matrix1.getM(); i += 3)
        {
            for (int j = 0; j < matrix2.getN(); j++)
            {
                double sum = 0;
                for (int k = 0; k < matrix1.getN(); k++)
                {
                    sum += matrix1.getMatrix(i, k) *
matrix2.getMatrix(k, j);
                }
                result.setMatrix(i, j, sum);
            }
        }
    }

    public void thirdThread()
    {
        for (int i = 2; i < matrix1.getM(); i += 3)
        {
            for (int j = 0; j < matrix2.getN(); j++)
            {
                double sum = 0;
```

```
                for (int k = 0; k < matrix1.getN(); k++)
                {
                    sum += matrix1.getMatrix(i, k) *
matrix2.getMatrix(k, j);
                }
                result.setMatrix(i, j, sum);
            }
        }
    }
}

class FourThread implements Runnable
{
    Matrix matrix1, matrix2, result;

    public FourThread(Matrix matrix1, Matrix matrix2, Matrix result)
    {
        this.matrix1 = matrix1;
        this.matrix2 = matrix2;
        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 < matrix1.getM(); i += 4)
        {
            for (int j = 0; j < matrix2.getN(); j++)
            {
                double sum = 0;
                for (int k = 0; k < matrix1.getN(); k++)
                {
                    sum += matrix1.getMatrix(i, k) *
matrix2.getMatrix(k, j);
                }
                result.setMatrix(i, j, sum);
            }
        }
    }

    public void secondThread()
    {
        for (int i = 1; i < matrix1.getM(); i += 4)
        {
            for (int j = 0; j < matrix2.getN(); j++)
            {
                double sum = 0;
                for (int k = 0; k < matrix1.getN(); k++)
                {
                    sum += matrix1.getMatrix(i, k) *
matrix2.getMatrix(k, j);
                }
                result.setMatrix(i, j, sum);
            }
        }
    }

    public void thirdThread()
    {
        for (int i = 2; i < matrix1.getM(); i += 4)
        {
            for (int j = 0; j < matrix2.getN(); j++)
            {
                double sum = 0;
                for (int k = 0; k < matrix1.getN(); k++)
                {
                    sum += matrix1.getMatrix(i, k) *
matrix2.getMatrix(k, j);
```

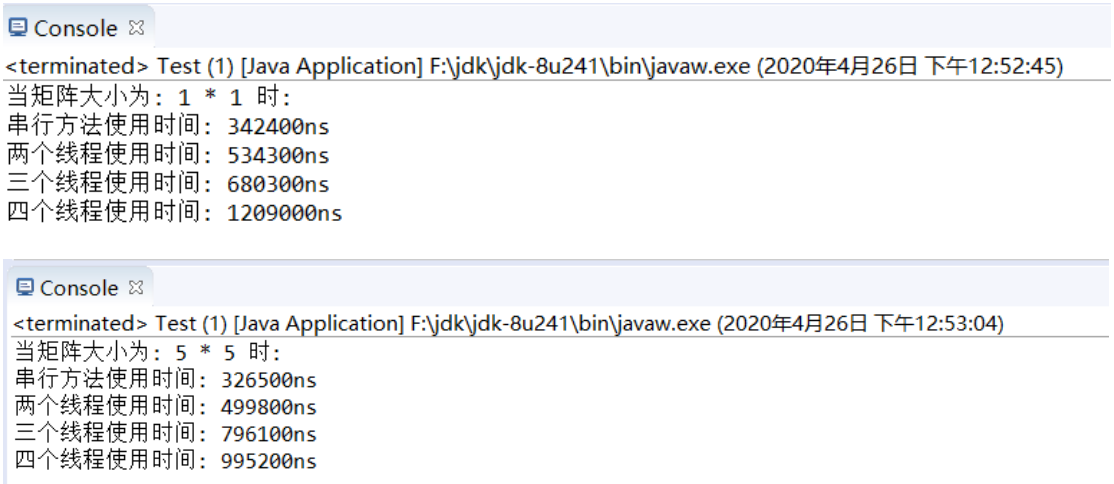
```
                }
                result.setMatrix(i, j, sum);
            }
        }
    }

    public void fourthThread()
    {
        for (int i = 3; i < matrix1.getM(); i += 4)
        {
            for (int j = 0; j < matrix2.getN(); j++)
            {
                double sum = 0;
                for (int k = 0; k < matrix1.getN(); k++)
                {
                    sum += matrix1.getMatrix(i, k) *
matrix2.getMatrix(k, j);
                }
                result.setMatrix(i, j, sum);
            }
        }
    }
}
```

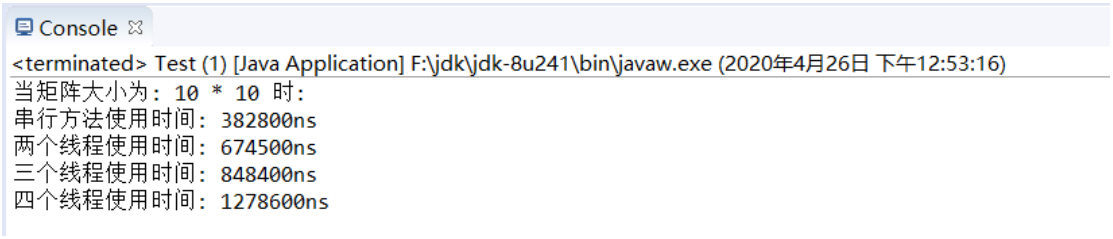
Test. java:

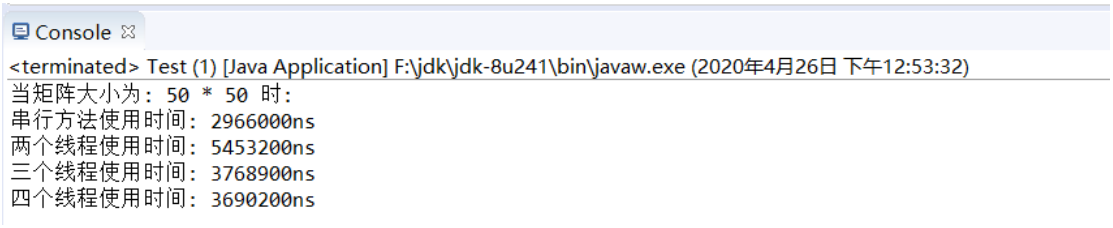
```
package xsy.lab4;

public class test
{
    public static void main(String[] args) throws
InterruptedException
    {
        int size = 50;

        Matrix matrix1 = new MatrixGenerator(size, size).getMatrix();
        Matrix matrix2 = new MatrixGenerator(size, size).getMatrix();

        long time1 = System.nanoTime();
        Matrix resultSequentially =
MatrixMultiplication.multiplySequentially(matrix1, matrix2);
        long time2 = System.nanoTime();
        Matrix resultParallelTwoThread =
MatrixMultiplication.multiplyParallelTwoThread(matrix1, matrix2);
```

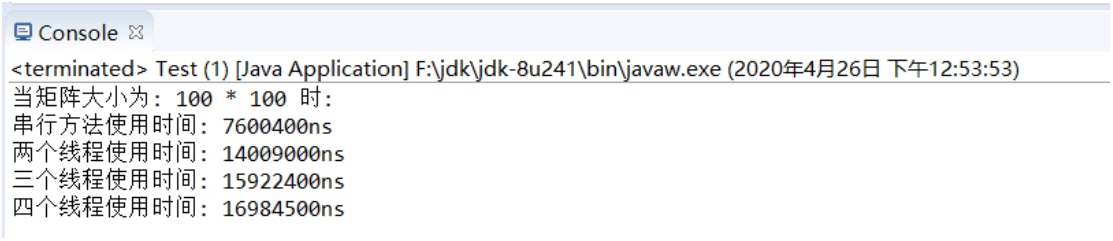
```
        long time3 = System.nanoTime();
        Matrix resultParallelThreeThread =
MatrixMultiplication.multiplyParallelThreeThread(matrix1, matrix2);
        long time4 = System.nanoTime();
        Matrix resultParallelFourThread =
MatrixMultiplication.multiplyParallelFourThread(matrix1, matrix2);
        long time5 = System.nanoTime();

        assert resultSequentially.equals(resultParallelTwoThread);
        assert resultSequentially.equals(resultParallelThreeThread);
        assert resultSequentially.equals(resultParallelFourThread);

        System.out.println("当矩阵大小为: " + size + " * " + size + "
时:");
        System.out.println("串行方法使用时间: " + (time2 - time1) +
"ns");
        System.out.println("两个线程使用时间: " + (time3 - time2) +
"ns");
        System.out.println("三个线程使用时间: " + (time4 - time3) +
"ns");
        System.out.println("四个线程使用时间: " + (time5 - time4) +
"ns");
    }
}
```

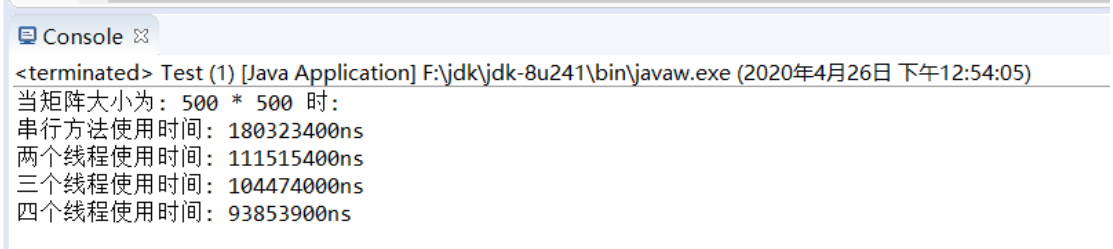
三、实验结果

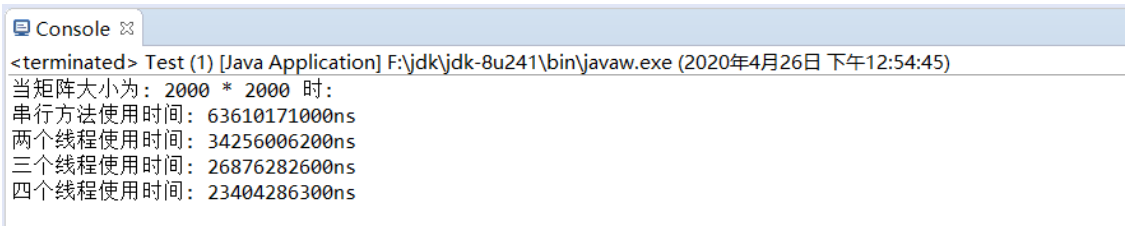




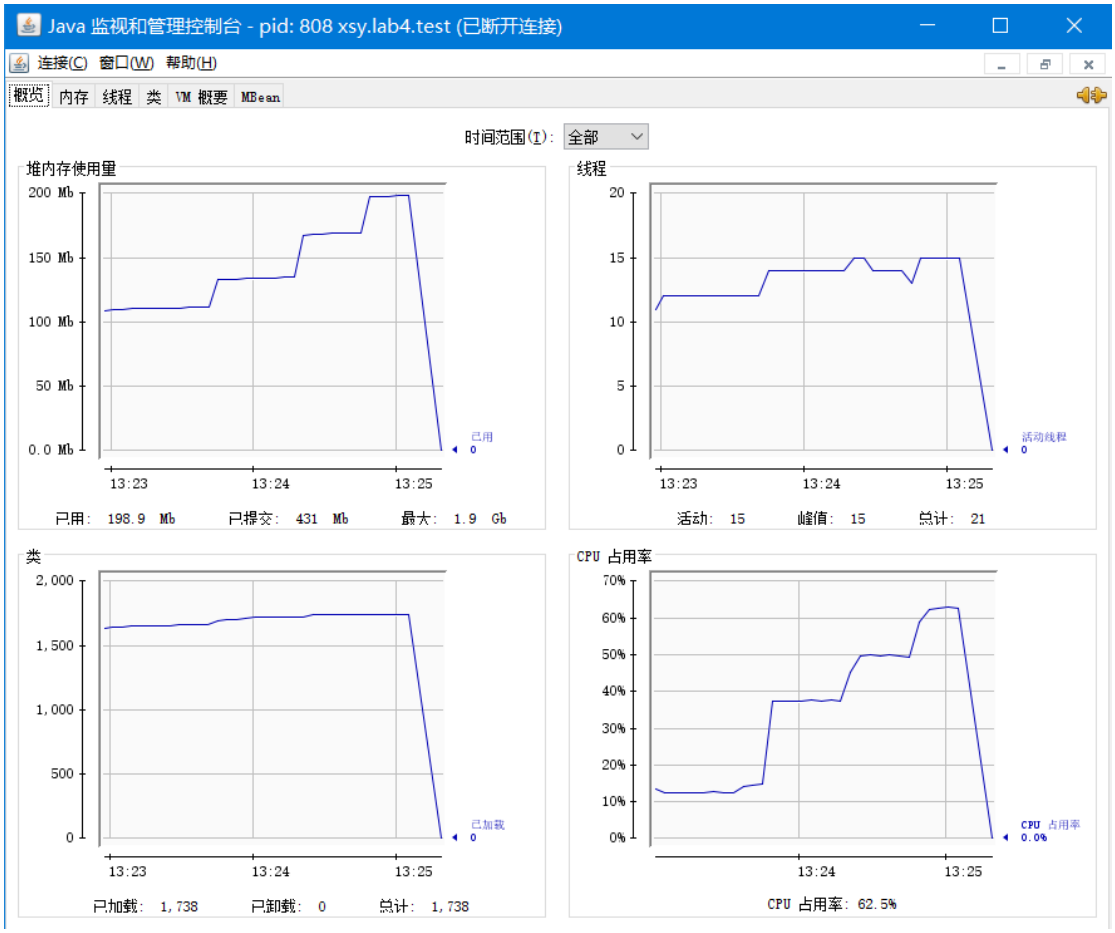








由上述结果可以大致得出结论：
当矩阵规模相对较小时，并行比串行效率低，并且效率随着线程数的增加而降低；
当矩阵规模相对较大时，并行比串行效率高，并且效率随着线程数的增加而升高。



根据矩阵大小为 2000*2000 时调用 java 监视与管理控制台结果分析可得：
当矩阵规模较大时，多线程并发方法会占用更多的堆内存、CPU 等资源，所以会使得乘法执行速度相较于串行方法更快；同时更多的线程数会占用更多的资源，所以效率会随着线程数的增加而升高。
而当矩阵规模较小时，多线程的方法相较于串行方法会有更多的线程创建与调度上的开销，而计算对 CPU 等资源的要求相对而言不是特别高，所以就会造成多线程的方法效率比串行方法低。