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　3018216235

姓　名　赵浩喆

一、实验内容

JAVA 进阶第四次实验：矩阵相乘

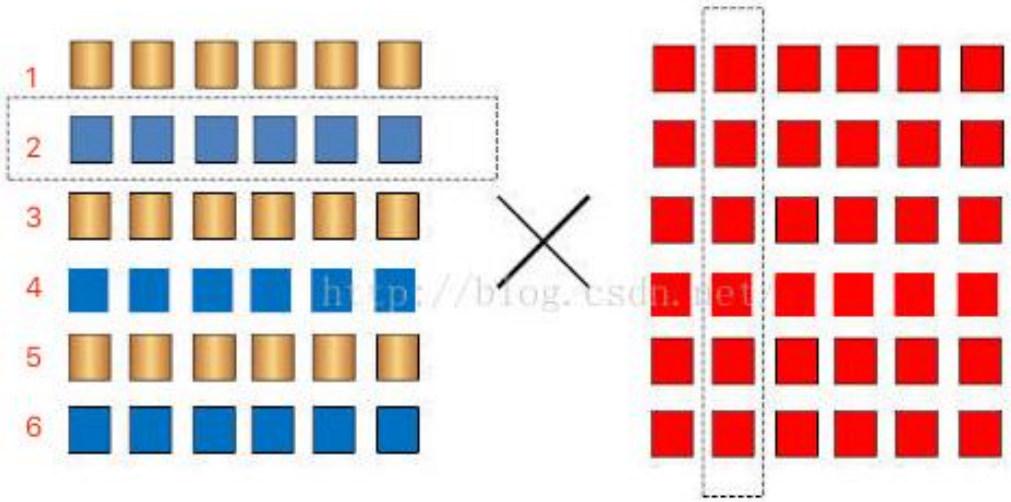
第四次实验是使用多线程编程技术，编写矩阵乘法。

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。

说明

矩阵乘法的方式不再赘述，由于矩阵乘法具有独立性，故可以使用多个线程来分别计算。



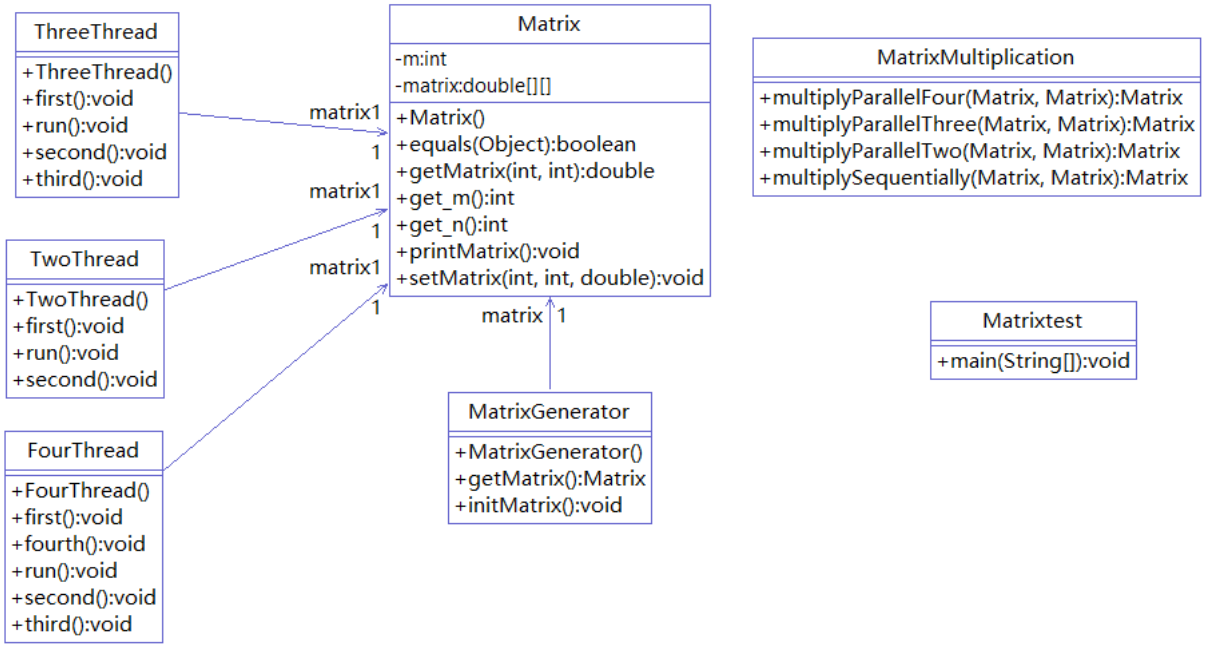
如图，图中的矩阵 1 可以分块成黄色和蓝色两部分，黄色：1， 3， 5。蓝色：2、4、6。于是我们可以使用两个线程对两种颜色分别计算，最后合并成一个结果。

分块的方式有很多种，这里可以按照行分，也可以按照列分，也可以分成四个 3*3 的子矩阵。

实验中需要大家分析不同的矩阵大小，不同的线程数，其时间产生的影响，同时也要保证结果的正确性，可以使用串行方法的结果作为标准，与多线程的方法进行比较。（可以使用断言进行判断：`assert func1.res == func2.res;`，断言开启方式在 VM options 加上 -ea 即可）

注意，生成的两个矩阵相乘需要有意义（a * b · b * c）

二、UML 图：



三、源代码：

```java
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 get_m()
{
    return m;
}

public int get_n()
{
    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.get_m() || this.n != c.get_n())
            {
                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;
```

```
            }
        }
    }
}
```

```
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.get_m(); i++)
        {
            for (int j = 0; j < this.matrix.get_n(); j++)
            {
                this.matrix.setMatrix(i, j, r.nextInt(100));
            }
        }
    }

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

```
public class MatrixMultiplication
{
    public static Matrix multiplySequentially(Matrix x, Matrix y)//串行
    {
        int a = x.get_m();
```

```
        int b1 = x.get_n();
        int b2 = y.get_m();
        int c = y.get_n();
        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 multiplyParallelTwo(Matrix x, Matrix y) throws
InterruptedException//二线程
    {
        int a = x.get_m();
        int b1 = x.get_n();
        int b2 = y.get_m();
        int c = y.get_n();
        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 multiplyParallelThree(Matrix x, Matrix y) throws
InterruptedException//三线程
    {
        int a = x.get_m();
        int b1 = x.get_n();
        int b2 = y.get_m();
        int c = y.get_n();
        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 multiplyParallelFour(Matrix x, Matrix y) throws
InterruptedException//四线程
    {
        int a = x.get_m();
        int b1 = x.get_n();
        int b2 = y.get_m();
```

```
        int c = y.get_n();
        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"))
        {
```

```
            first();
        }
        else if (Thread.currentThread().getName().equals("线程2"))
        {
            second();
        }
    }

    public void first()
    {
        for (int i = 0; i < matrix1.get_m(); i += 2)
        {
            for (int j = 0; j < matrix2.get_n(); j++)
            {
                double sum = 0;
                for (int k = 0; k < matrix1.get_n(); k++)
                {
                    sum += matrix1.getMatrix(i, k) * matrix2.getMatrix(k,
j);
                }
                result.setMatrix(i, j, sum);
            }
        }

    }

    public void second()
    {
        for (int i = 1; i < matrix1.get_m(); i += 2)
        {
            for (int j = 0; j < matrix2.get_n(); j++)
            {
                double sum = 0;
                for (int k = 0; k < matrix1.get_n(); 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"))
        {
            first();
        }
        else if (Thread.currentThread().getName().equals("线程2"))
        {
            second();
        }
        else if (Thread.currentThread().getName().equals("线程3"))
        {
            third();
        }
    }

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

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

    public void third()
    {
        for (int i = 2; i < matrix1.get_m(); i += 3)
        {
            for (int j = 0; j < matrix2.get_n(); j++)
            {
                double sum = 0;
                for (int k = 0; k < matrix1.get_n(); 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"))
        {
            first();
        }
        else if (Thread.currentThread().getName().equals("线程2"))
        {
            second();
        }
        else if (Thread.currentThread().getName().equals("线程3"))
        {
            third();
        }
        else if (Thread.currentThread().getName().equals("线程4"))
        {
            fourth();
        }
    }

    public void first()
    {
        for (int i = 0; i < matrix1.get_m(); i += 4)
        {
            for (int j = 0; j < matrix2.get_n(); j++)
            {
                double sum = 0;
                for (int k = 0; k < matrix1.get_n(); k++)
                {
                    sum += matrix1.getMatrix(i, k) * matrix2.getMatrix(k,
j);
                }
                result.setMatrix(i, j, sum);
            }
        }
    }

    public void second()
    {
        for (int i = 1; i < matrix1.get_m(); i += 4)
```

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

    public void third()
    {
        for (int i = 2; i < matrix1.get_m(); i += 4)
        {
            for (int j = 0; j < matrix2.get_n(); j++)
            {
                double sum = 0;
                for (int k = 0; k < matrix1.get_n(); k++)
                {
                    sum += matrix1.getMatrix(i, k) * matrix2.getMatrix(k,
j);
                }
                result.setMatrix(i, j, sum);
            }
        }
    }

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

```
            }
        }
    }
}
```

```
public class Matrixtest
{
    public static void main(String[] args) throws InterruptedException
    {
        int size = 20;

        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.multiplyParallelTwo(matrix1, matrix2);

        long time3 = System.nanoTime();

        Matrix resultParallelThreeThread =
MatrixMultiplication.multiplyParallelThree(matrix1, matrix2);

        long time4 = System.nanoTime();

        Matrix resultParallelFourThread =
MatrixMultiplication.multiplyParallelFour(matrix1, matrix2);

        long time5 = System.nanoTime();

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

    System.out.println("==============================================");
```

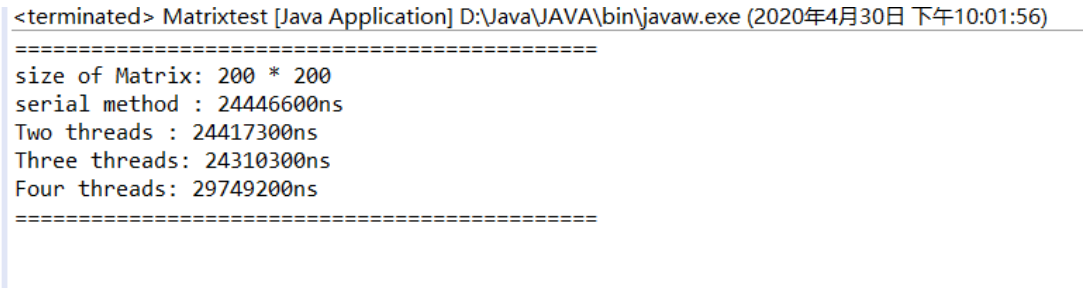
```
        System.out.print("size of Matrix: " + size + " * " + size +"\n");
        System.out.print("serial method : " + (time2 - time1) + "ns\n");
        System.out.print("Two threads : " + (time3 - time2) + "ns\n");
        System.out.print("Three threads: " + (time4 - time3) + "ns\n");
        System.out.print("Four threads: " + (time5 - time4) + "ns\n");

    System.out.println("=============================================");
    }
}
```

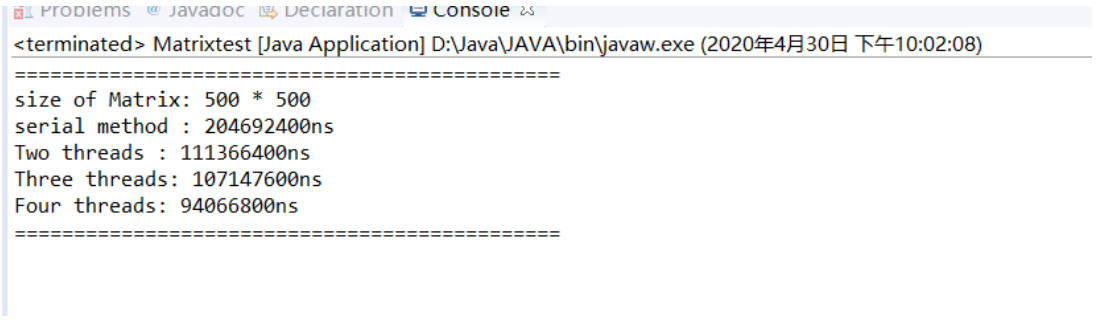
四、实验结果：

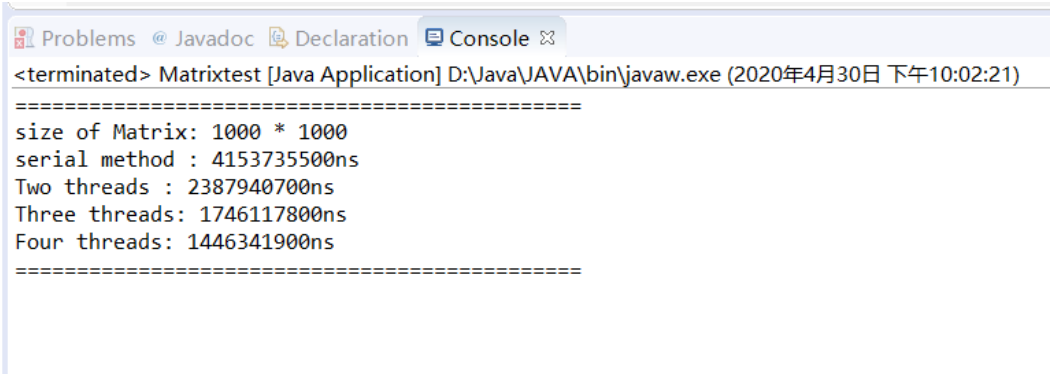
```
<terminated> Matrixtest [Java Application] D:\Java\JAVA\bin\javaw.exe (2020年4月30日 下午10:01:21)
=============================================
size of Matrix: 5 * 5
serial method : 600200ns
Two threads : 687600ns
Three threads: 794600ns
Four threads: 947400ns
=============================================
```

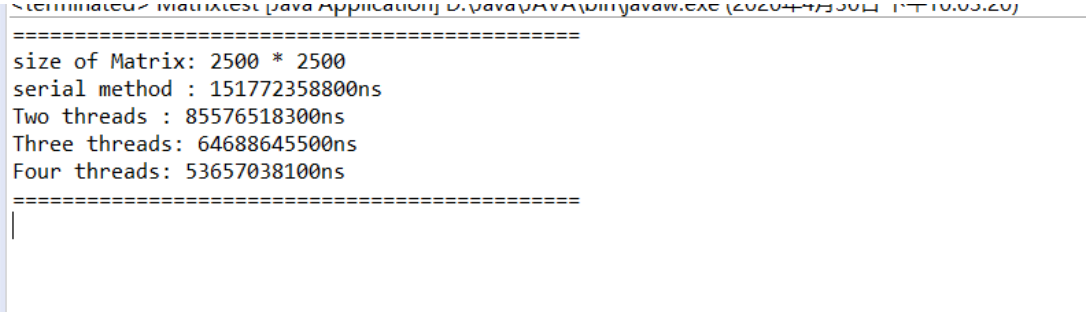
```
<terminated> test [Java Application] D:\Java\JAVA\bin\javaw.exe (2020年4月30日 下午9:56:10)
=============================================
size of Matrix: 20 * 20
serial method : 796800ns
Two threads : 946300ns
Three threads: 817200ns
Four threads: 876900ns
=============================================
```

```
<terminated> Matrixtest [Java Application] D:\Java\JAVA\bin\javaw.exe (2020年4月30日 下午10:01:42)
=============================================
size of Matrix: 50 * 50
serial method : 3559500ns
Two threads : 6476600ns
Three threads: 3876000ns
Four threads: 4376700ns
=============================================
```









五、结果分析

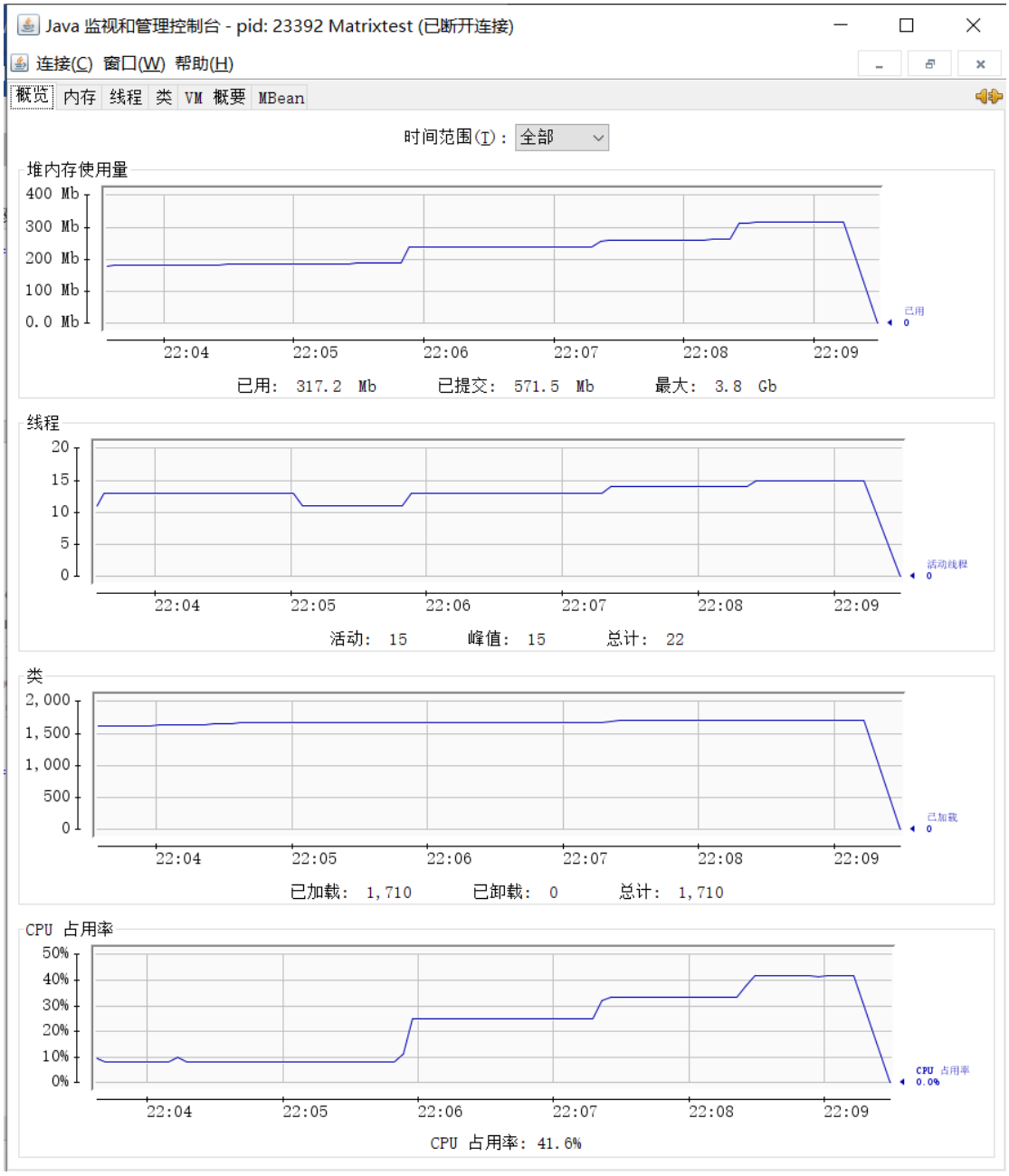
依次将矩阵按照：5*5, 20*20, 50*50 ,200*200, 500*500, 2500*2500 的大小作为样例进行测试。可知：

- 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。

调用 java 监视与管理控制台进行结果分析如下，以矩阵样例大小为 2500*2500 为例，可知当矩阵规模较大时，多线程并发方法会占用更多的堆内存、CPU 等资源，所以这虽然使得乘法执行速度相较于串行方法更快，但会占用更多的资源。程序运行的效率随着线程数的增加而升高。表现为并行方法使用时间相较于串行方法而言越来越短。

而对于规模较小的矩阵，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，同时较小矩阵的计算对 CPU 等资源的要求相对而言不是特别高，所以使得多线程的方法效率比串行方法低。表现为串行方法使用时间较短。


Java 监视和管理控制台 - pid: 23392 Matrixtest (已断开连接)
连接(C) 窗口(W) 帮助(H)
概览 内存 线程 类 VM 概要 MBean
时间范围(T)： 全部
堆内存使用量
400 Mb
300 Mb
200 Mb
100 Mb
0.0 Mb
22:04 22:05 22:06 22:07 22:08 22:09
已用
0
已用： 317.2 Mb 已提交： 571.5 Mb 最大： 3.8 Gb
线程
20
15
10
5
0
22:04 22:05 22:06 22:07 22:08 22:09
活动线程
0
活动： 15 峰值： 15 总计： 22
类
2,000
1,500
1,000
500
0
22:04 22:05 22:06 22:07 22:08 22:09
已加载
0
已加载： 1,710 已卸载： 0 总计： 1,710
CPU 占用率
50%
40%
30%
20%
10%
0%
22:04 22:05 22:06 22:07 22:08 22:09
CPU 占用率
0.0%
CPU 占用率： 41.6%