JAVA 编程进阶上机报告

Lab 4 Multithreading and Matrix Multiplication

学　　院　智能与计算学部

专　　业　软件工程

姓　　名　张瑞安

学　　号　3018218063

年　　级　2018

班　　级　4

一、实验目的

使用多线程技术，编写矩阵乘法

二、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。

三、设计思路及源码

- 随机生成矩阵，且要作矩阵乘法，需要生成2个矩阵且遵循 `m*p` 、 `n*p` 原则，分别使用 `List` 存储。

  `MatrixGenerator.java`

  ```java
  import java.util.ArrayList;
  import java.util.List;
  import java.util.Random;

  public class MatrixGenerator {

      private List<List<Double>> matrix_1 = new ArrayList<>();
      private List<List<Double>> matrix_2 = new ArrayList<>();

      //m行p列，p行n列
      public MatrixGenerator(int max){
          int m = new Random().nextInt(max);
          int p = new Random().nextInt(max);
          int n = new Random().nextInt(max);
          System.out.println("m="+m+"\np="+p+"\nn="+n);
          for (int i=0;i<m;i++){
              this.matrix_1.add(
                      LineGen(p)
              );
          }
          for (int i=0;i<p;i++){
              this.matrix_2.add(
                      LineGen(n)
              );
          }
      }

      public List<Double> LineGen(int width){
          List<Double> line = new ArrayList<>();
  ```

```
        for (int i=0;i<width;i++){
            line.add(new Random().nextDouble());
        }
        return line;
    }

    public List<List<Double>> getMatrix_1() {
        return matrix_1;
    }

    public List<List<Double>> getMatrix_2() {
        return matrix_2;
    }
}
```

- 串行乘法，将矩阵 `m1` 中一行与 `m2` 中一列进行提取计算得到对应结果矩阵其中一个元素的值，推导出 `m1` 中一行与 `m2` 中所有列进行计算得到对应结果矩阵其中一行的所有值，从而推导出结果矩阵的算法。

  `NormalMultiplication.java`

  ```
  import java.util.ArrayList;
  import java.util.List;

  public class NormalMultiplication {

      public List<List<Double>> Multiply(List<List<Double>> m1, List<List<Double>> m2){
          int m = m1.size();
          List<List<Double>> res = new ArrayList<>();
          for (int i=0;i<m;i++){
              res.add(singleLine(m1.get(i),m2));
          }
          return res;
      }

      public List<Double> singleLine(List<Double> line ,List<List<Double>> m2) {
          List<Double> res = new ArrayList<>();
          int p = m2.size();
          int n = m2.get(0).size();
          for (int i=0;i<n;i++){
              List<Double> col = new ArrayList<>();
              for (int j=0;j<p;j++){
                  col.add(m2.get(j).get(i));
              }
              res.add(singleAtom(line,col));
          }
          return res;
      }

      public double singleAtom(List<Double> line, List<Double> column){
          double res=0;
          for (int i=0;i<line.size();i++){
              res+=line.get(i)*column.get(i);
          }
          return res;
  ```

```
    }
}
```

- 并行乘法，将矩阵 `m1` 动态按行分割为若干小矩阵进行并行计算，算法同上。

  `MultithreadingMultiplication.java`

```java
import java.util.ArrayList;
import java.util.List;
import java.util.concurrent.CountDownLatch;

public class MultithreadingMultiplication extends Thread{

    private List<List<Double>> m1;
    private List<List<Double>> m2;
    private int index;
    private int gap;
    private CountDownLatch countDownLatch;
    private List<List<Double>>[] res;

    public MultithreadingMultiplication(
            List<List<Double>> m1, List<List<Double>> m2, int index, int
gap, CountDownLatch countDownLatch, List<List<Double>>[] res
    ){
        this.m1=m1;
        this.m2=m2;
        this.index=index;
        this.gap=gap;
        this.res=res;
        this.countDownLatch=countDownLatch;
    }

    public void run(){
        List<List<Double>> re = new ArrayList<>();
        for (int i = index*gap;i<(index+1)*gap;i++){
            re.add(new NormalMultiplication().singleLine(m1.get(i),m2));
        }
        res[index] = re;
        countDownLatch.countDown();
    }

}
```

- 测试代码

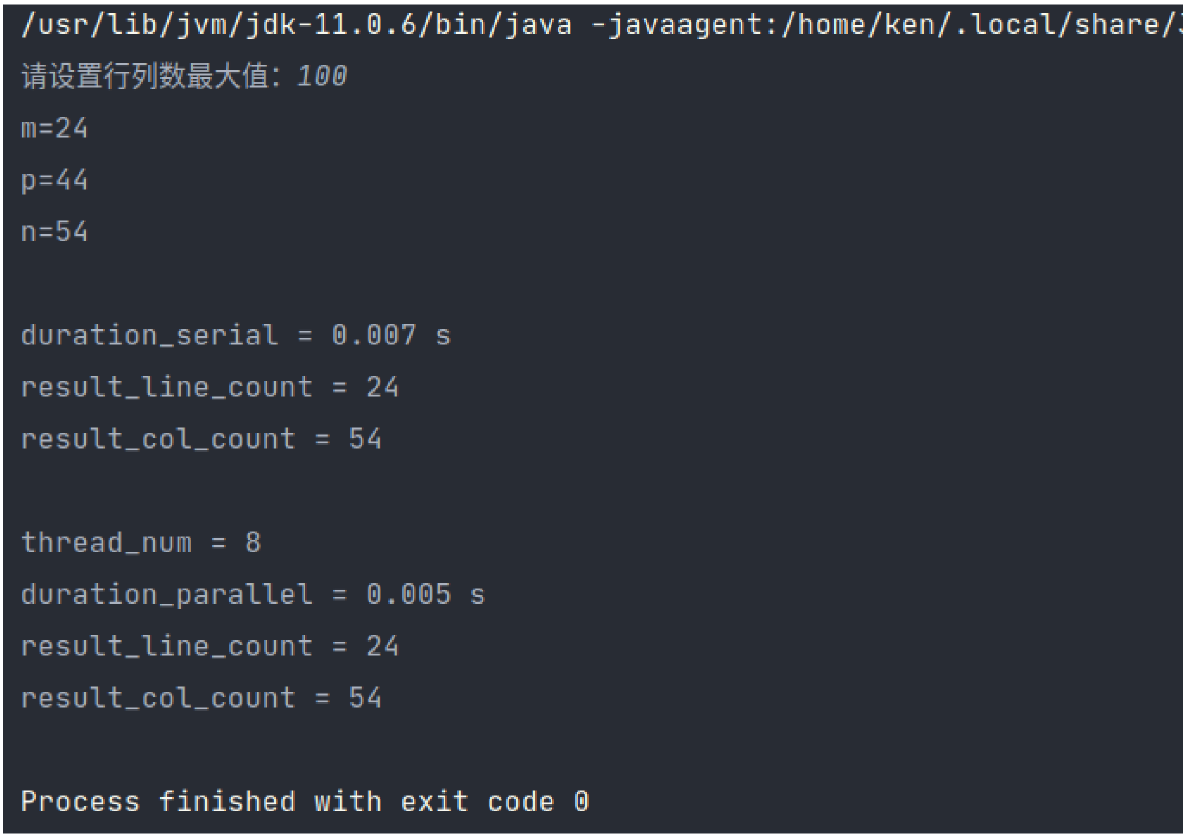
  `Test.java`

```java
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;
import java.util.concurrent.CountDownLatch;

public class Test {

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

        System.out.print("请设置行列数最大值：");
        Scanner scanner = new Scanner(System.in);
        int max = scanner.nextInt();
        MatrixGenerator matrixGenerator = new MatrixGenerator(max);
        long started_at = System.currentTimeMillis();
        List<List<Double>> res = new
NormalMultiplication().Multiply(matrixGenerator.getMatrix_1(),matrixGenerato
r.getMatrix_2());
        System.out.println("\nduration_serial = "+
((System.currentTimeMillis() - started_at)/1000.0)+" s"+
                "\nresult_line_count = "+res.size()+
                "\nresult_col_count = "+res.get(0).size());

        int threadnum = 0;
        CountDownLatch countDownLatch = new CountDownLatch(threadnum);
        int m1_size = matrixGenerator.getMatrix_1().size();
        for (int i=1;i<10;i++){
            if (m1_size%i == 0){
                threadnum = i;
            }
        }
        System.out.println("\nthread_num = "+threadnum);
        int gap = m1_size / threadnum;
        List<List<Double>>[] result = new List[threadnum];
        started_at = System.currentTimeMillis();
        for (int i=0;i<threadnum;i++){
            MultithreadingMultiplication multithreadingMultiplication = new
MultithreadingMultiplication(

 matrixGenerator.getMatrix_1(),matrixGenerator.getMatrix_2(),i,gap,countDown
Latch,result
            );
            multithreadingMultiplication.start();
        }
        countDownLatch.await();

        System.out.println(
                "duration_parallel = "+((System.currentTimeMillis()-
started_at)/1000.0)+" s"+
                        "\nresult_line_count = "+
(result.length*result[0].size()) +
                        "\nresult_col_count = "+result[0].get(0).size()

        );

    }

}
```

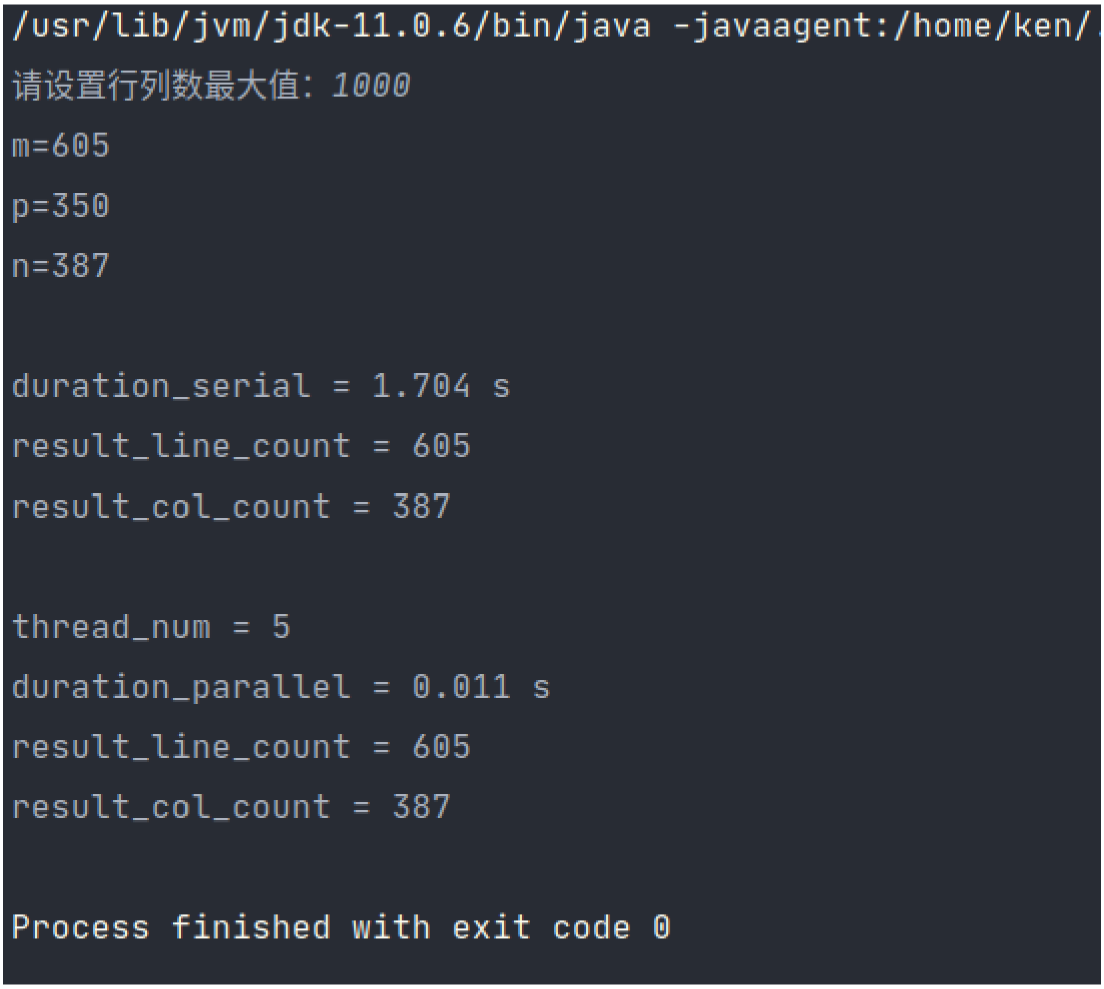
四、测试结果

> 注:
>
> 1、行列数最大值max为生成矩阵行列数在 `[0,max]` 之间取值
>
> 2、时间记录方式为开始计算时间减去结束计算时间

> 3、第一部分为两个矩阵行列数，第二个部分为串行计算结果，第三部分为并行计算结果（后两个部分包括时间和结果矩阵行列数）



```
/usr/lib/jvm/jdk-11.0.6/bin/java -javaagent:/home/ken/.local/share/
请设置行列数最大值：100
m=24
p=44
n=54

duration_serial = 0.007 s
result_line_count = 24
result_col_count = 54

thread_num = 8
duration_parallel = 0.005 s
result_line_count = 24
result_col_count = 54

Process finished with exit code 0
```



```
/usr/lib/jvm/jdk-11.0.6/bin/java -javaagent:/home/ken/
请设置行列数最大值：1000
m=605
p=350
n=387

duration_serial = 1.704 s
result_line_count = 605
result_col_count = 387

thread_num = 5
duration_parallel = 0.011 s
result_line_count = 605
result_col_count = 387

Process finished with exit code 0
```