**中山大学计算机院本科生实验报告**

**（2024学年春季学期）**

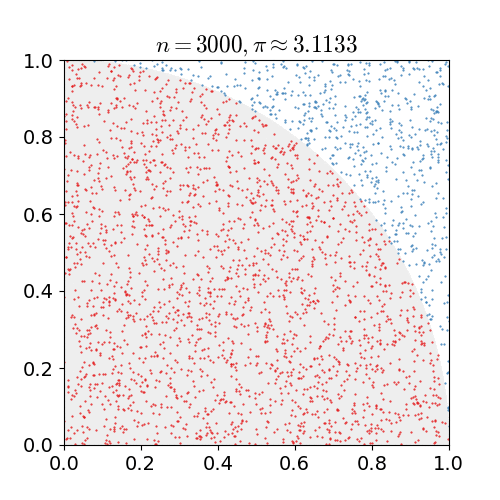
**课程名称：并行程序设计 批改人：**

|  |  |  |  |
| --- | --- | --- | --- |
| **实验** | **4-Pthreads并行方程求解及蒙特卡洛** | **专业（方向）** | **计算机科学与技术** |
| **学号** | **21307174** | **姓名** | **刘俊杰** |
| **Email** | **liujj255@mail2.sysu.edu.cn** | **完成日期** | **2024/4/15** |

# **实验目的**

**蒙特卡洛方法求的近似值**

**基于Pthreads编写多线程程序，使用蒙特卡洛方法求圆周率近似值。**

****

**蒙特卡洛方法与圆周率近似：蒙特卡洛方法是一种基于随机采样的数值计算方法，通过模拟随机时间的发生，来解决各类数学、物理和工程上的问题，尤其是直接解析解决困难或无法求解的问题。其基本思想是：当问题的确切解析解难以获得时，可以通过随机采样的方式，生成大量的模拟数据，然后利用这些数据的统计特性来近似求解问题。在计算圆周率值时，可以随机地将点撒在一个正方形内。当点足够多时（见上图），总采样点数量与落在内切圆内采样点数量的比例将趋近于，可据此来估计的值。**

**输入：整数，取值范围为[1024, 65536]**

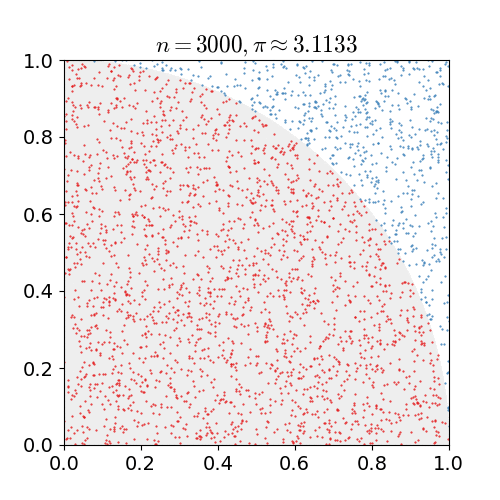
**问题描述：随机生成正方形内的个采样点，并据此估算的值。**

**输出：总点数，落在内切圆内点数，估算的值，及消耗的时间。**

**要求：基于Pthreads编写多线程程序，使用蒙特卡洛方法求圆周率近似值。讨论程序并行性能。**

# **实验过程和核心代码**

## **2.1实验思路**

****

**蒙特卡洛方法与圆周率近似：蒙特卡洛方法是一种基于随机采样的数值计算方法，通过模拟随机时间的发生，来解决各类数学、物理和工程上的问题，尤其是直接解析解决困难或无法求解的问题。其基本思想是：当问题的确切解析解难以获得时，可以通过随机采样的方式，生成大量的模拟数据，然后利用这些数据的统计特性来近似求解问题。在计算圆周率值时，可以随机地将点撒在一个正方形内。当点足够多时（见上图），总采样点数量与落在内切圆内采样点数量的比例将趋近于，可据此来估计的值。**

## **2.2 实验过程**

**①接收运行参数中的线程数数量和产生随机点的个数。**

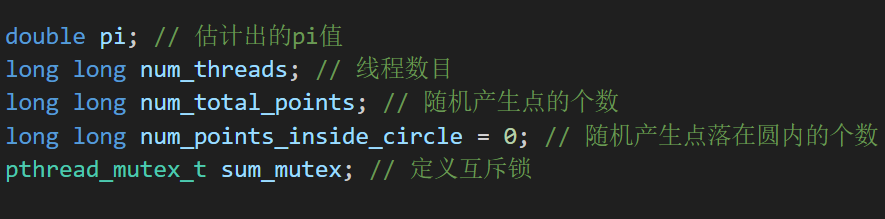
**②创建线程，每个线程都运行函数。**

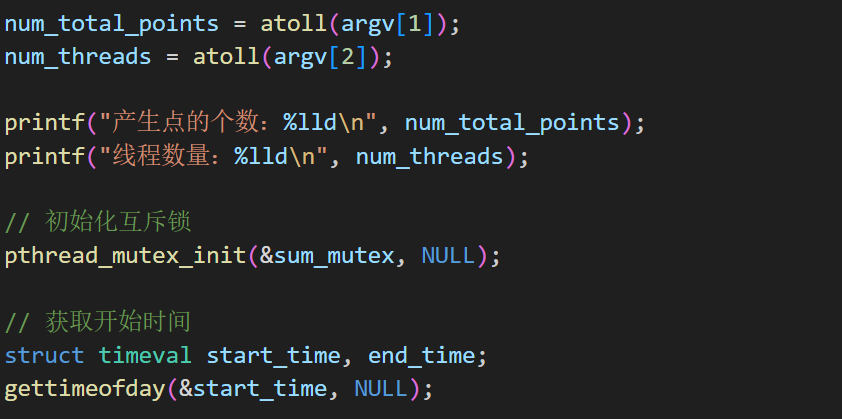
**③每个线程计算自己负责计算随机点的个数，随机生成 在矩形中的点，利用其与原点的距离判断是否在圆中，如果在则统计圆中点数的数字加1(利用互斥锁解决线程的竞争问题)。**

**④根据公式和随机点在圆中的概率计算出Π值，输出计算时间和误差。**

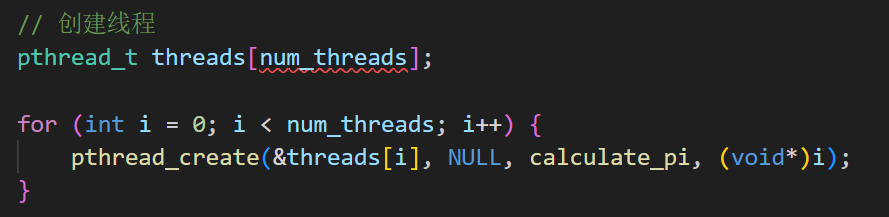
## **2.3 核心代码**

**①接收运行参数中的线程数数量和产生随机点的个数。**

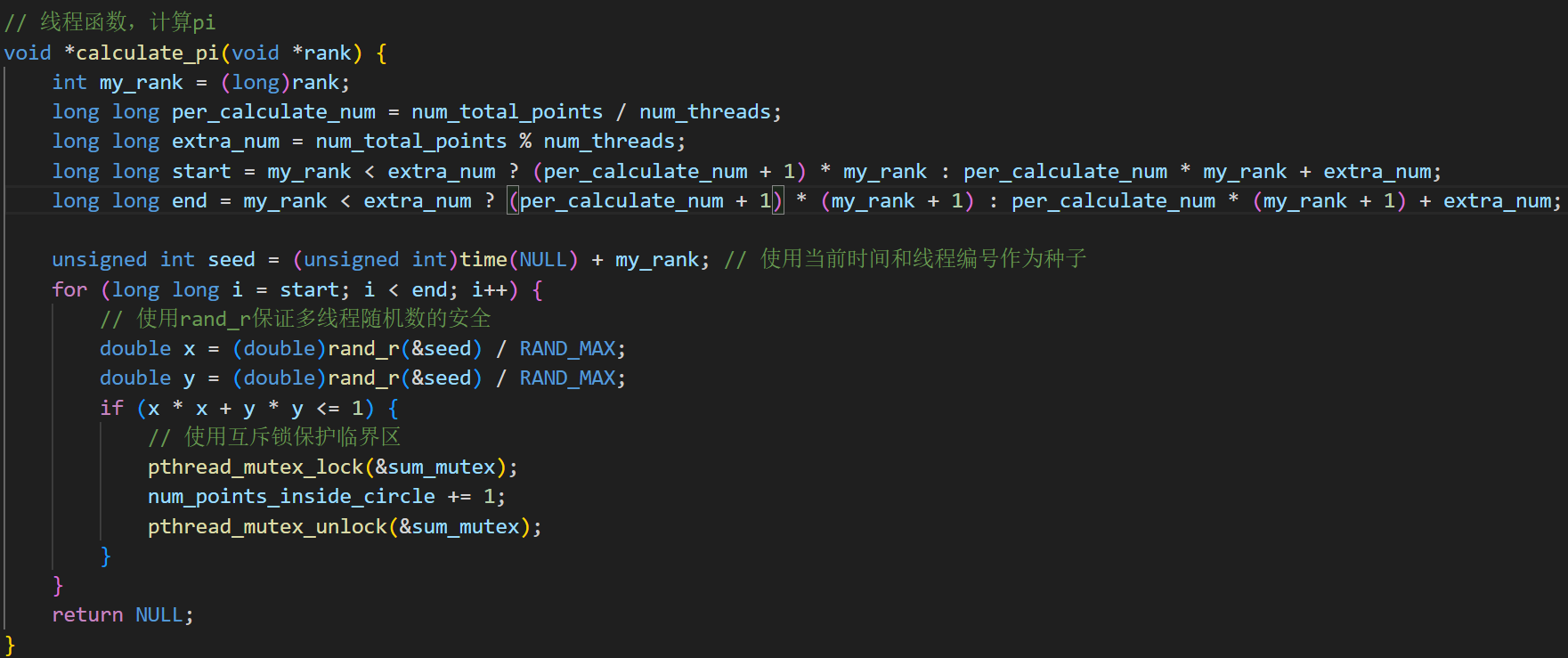




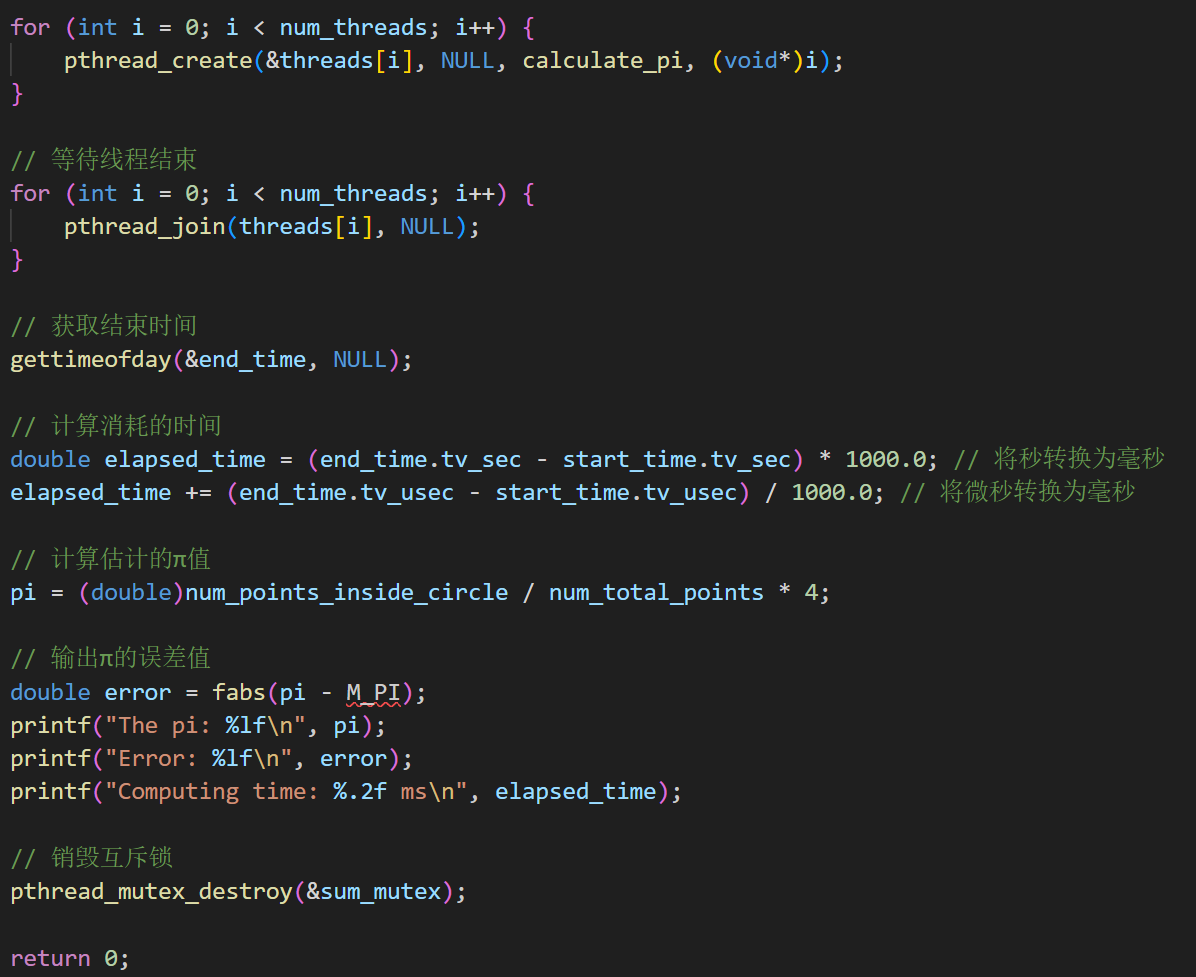
**②创建线程，每个线程都运行函数。**



**③每个线程计算自己负责计算随机点的个数，随机生成 在矩形中的点，利用其与原点的距离判断是否在圆中，如果在则统计圆中点数的数字加1(利用互斥锁解决线程的竞争问题)。**

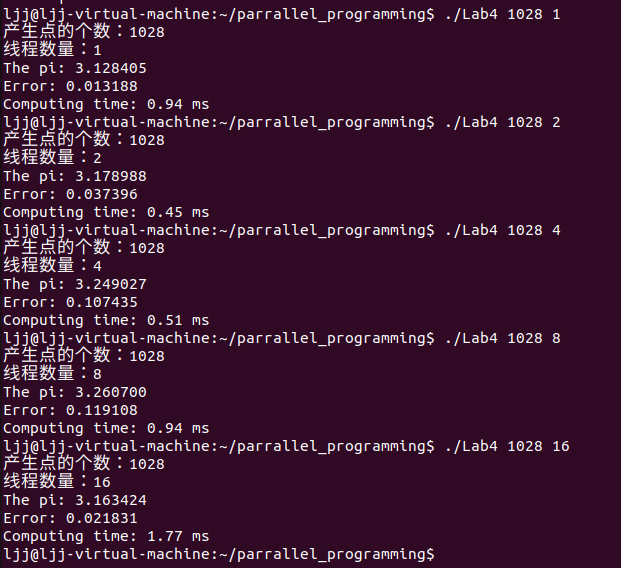


**④根据公式和随机点在圆中的概率计算出Π值，输出计算时间和误差。**



# **实验结果**

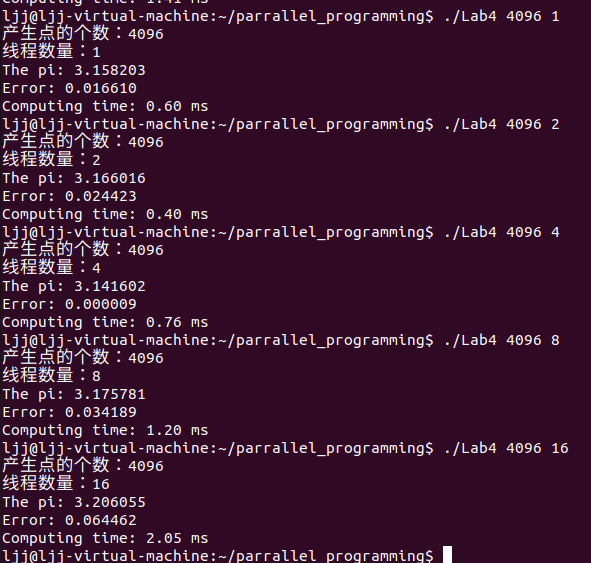
## **随机点个数为1024**

****

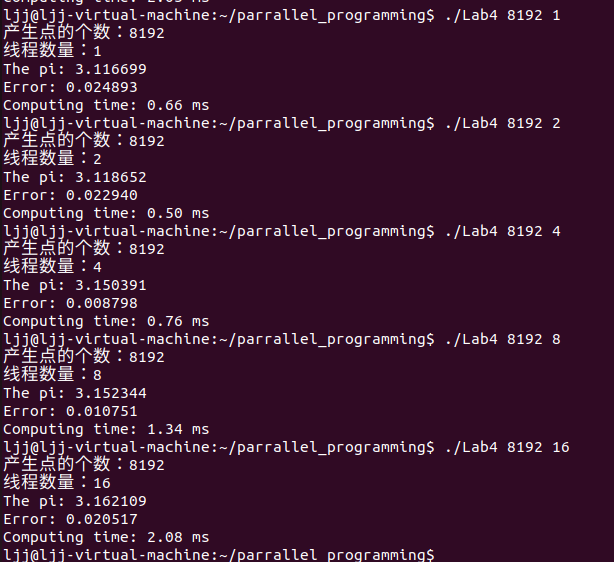
## **随机点个数为2048**

## 

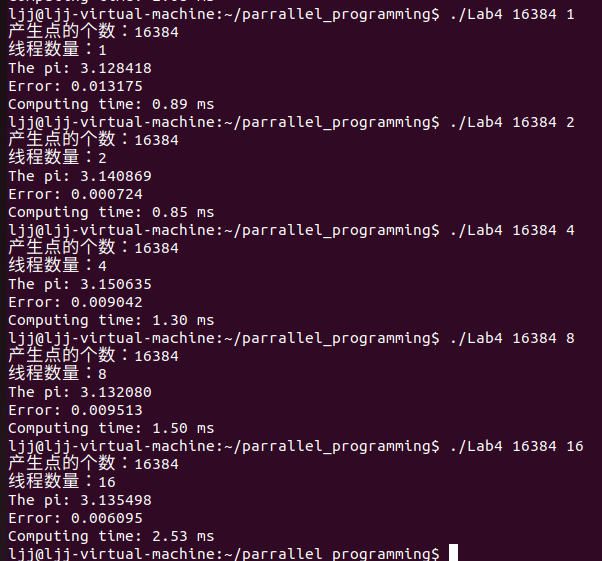
## **随机点个数为4096**

****

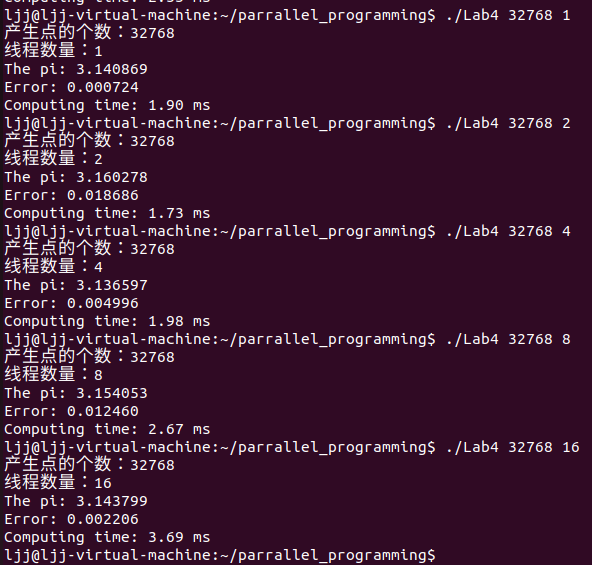
## **随机点个数为8192**

****

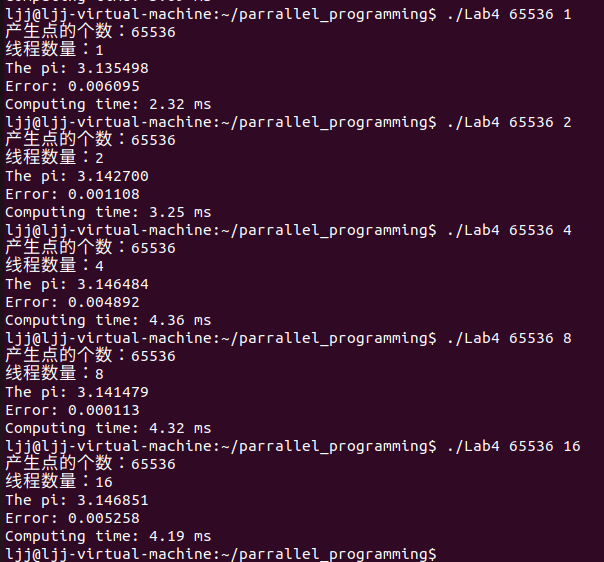
## **随机点个数为16384**

****

## **随机点个数为32768**

****

## **随机点个数为65536**

****

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **线程数** | **随机点数量** | | | | | | |
| **1024** | **2048** | **4096** | **8192** | **16384** | **32768** | **65536** |
| **1** | **3.128405**  **0.013188**  **0.94ms** | **3.107422**  **0.034171**  **0.40ms** | **3.158203**  **0.016610**  **0.60ms** | **3.116699**  **0.024893**  **0.66ms** | **3.128418**  **0.013175**  **0.89ms** | **3.140869**  **0.000724**  **1.90ms** | **3.135498**  **0.006095**  **2.32ms** |
| **2** | **3.178988**  **0.03736**  **0.45ms** | **3.08781**  **0.053702**  **0.36ms** | **3.166016**  **0.024423**  **0.40ms** | **3.118652**  **0.022940**  **0.50ms** | **3.140869**  **0.000724**  **0.85ms** | **3.160278**  **0.018686**  **1.73ms** | **3.142700**  **0.001108**  **3.25ms** |
| **4** | **3.249027**  **0.107438**  **0.51ms** | **3.117188**  **0.024405**  **0.58ms** | **3.141602**  **0.000009**  **0.76ms** | **3.150391**  **0.008798**  **0.76ms** | **3.150635**  **0.009042**  **1.30ms** | **3.136597**  **0.004996**  **1.98ms** | **3.149484**  **0.004892**  **4.36ms** |
| **8** | **3.260700**  **0.199108**  **0.94ms** | **3.199219**  **0.057626**  **1.03ms** | **3.175781**  **0.034189**  **1.20ms** | **3.152344**  **0,010751**  **1.34ms** | **3.132080**  **0.009513**  **1.50ms** | **3.154053**  **0.12460**  **2.67ms** | **3.141479**  **0.000113**  **4..32ms** |
| **16** | **3.163424**  **0.021831**  **1.77ms** | **3.121094**  **0.020499**  **1.41ms** | **3.206055**  **0.064462**  **2.06ms** | **3.162109**  **0.020517**  **2.08ms** | **3.135498**  **0.006095**  **2.53ms** | **3.143799**  **0.002206**  **3.69ms** | **3.146851**  **0.005258**  **4.19ms** |

# **实验感想**

## **4.1 问题:使用rand时产生的随机数都一样**

**解决方法：**

1. **使用time()和线程号作为种子**
2. **使用rand()**

**rand()和rand\_r的区别:**

**rand() 函数:**

**rand() 是C语言标准库中提供的伪随机数生成函数。**

**它使用一个全局状态来生成随机数。这个状态在多线程环境下是共享的，因此rand() 在多线程环境下不是线程安全的。**

**在多线程环境下，如果多个线程同时调用 rand() 函数，可能会导致竞态条件，使得生成的随机数不是预期的结果。**

**rand\_r() 函数:**

**rand\_r() 是 POSIX 标准中定义的伪随机数生成函数，具有可重入性（reentrant），也就是说，它可以安全地在多线程环境中使用。**

**rand\_r() 函数的使用方式不同于 rand()。它接受一个指向随机数生成器状态的指针作为参数，因此每个线程都可以有自己的状态，从而避免了竞态条件。**

**通过传递不同的状态指针，每个线程可以独立地生成随机数序列，而不会相互干扰。**

**为了在多线程环境中安全地生成随机数，应该使用 rand\_r() 而不是 rand()。通过使用 rand\_r()，可以确保每个线程都有自己的随机数生成器状态，从而避免了线程间的竞争和不确定性**

**4.2 实验感悟**

**①使用多线程可以加速计算过程，特别是在生成大量随机点的情况下。每个线程负责生成一部分点，并统计落在内切圆内的点数，从而提高计算效率。同时合理选择线程数目很重要，线程数目多使得估计的Π值越准确。**

**②在多线程环境中，需要特别注意随机数的生成。使用 rand\_r() 函数可以确保在多线程情况下生成的随机数是线程安全的，从而避免竞态条件。**

**③由于随机性是蒙特卡洛方法的本质，因此每次计算的结果可能会有一定的波动。通过多次运行程序并取平均值，可以提高估算结果的准确性。**