1 实验内容

1.1 实验意义

蒙特卡洛方法是一种基于随机采样的数值计算方法，广泛应用于统计物理、金融工程、机器学习等领域。π 值计算作为蒙特卡洛方法的经典应用，能够有效验证并行随机算法的正确性和性能。本实验通过 OpenMP 并行编程技术实现蒙特卡洛 π 值计算，探究 OpenMP 在随机数值计算中的性能表现和优化策略。

1.2 相关工作

蒙特卡洛方法求 π 值基于几何概率原理：在单位正方形内随机投点，统计落在四分之一圆内的点数比例。该方法的收敛速度为 $O(1/\sqrt{N})$，其中 N 为采样点数。相比确定性数值积分方法，蒙特卡洛方法具有维度无关性和易于并行化的优势，但需要大量采样点才能获得较高精度。

1.3 算法原理

基于几何概率原理：

1. 在单位正方形 $[0,1] \times [0,1]$ 内随机生成点 (x, y)
2. 判断点是否落在四分之一单位圆内：$\sqrt{x^2 + y^2} \leq 1$
3. 圆内点数与总点数的比值近似 $\pi/4$
4. 估算公式：$\pi \approx 4 \times \text{points\_in\_circle} / \text{total\_points}$

2 实验方法

2.1 并行化设计

任务划分策略：主要使用 OpenMP `parallel for` 指令并行化采样循环，通过 `private` 为每个线程独立生成随机数和判断点位置，采用 `reduction` 子句自动处理圆内点数累加，最后通过 `num_threads` 生成不同线程数下的计算结果。

随机数生成优化：初始方案使用 `rand()` 函数导致严重性能退化，若使用 `rand()` 并行计算会出现并行计算的时间远远大于串行计算，这是因为 `rand()` 函数的全局状态导致线程间竞争，频繁加锁解锁造成严重性能损失，并行效率甚至低于串行计算。通过线程安全的 `rand_r()` 函数替代 `rand()`，每个线程使用独立种子 `time(NULL) + thread_id + iterator_id`，实现真正的并行加速。

内存访问优化：声明私有变量避免 false sharing，使用 `reduction` 操作减少同步开销

算法优化：由于 $x^2 + y^2 \leq 1$ 等价于 $\sqrt{x^2 + y^2} \leq 1$，因此可以避免不必要的开平方运算，从而提高计算效率。

3　性能评估

3.1　实验环境

- **硬件配置**
  - CPU: Intel(R) Core(TM) i5-12600KF（10核16线程：6P-core @4.9GHz, 4E-core @3.6GHz）
  - Cache 缓存：L1 80KB/核（P-core），L2 1.25MB/核（P-core），L3 20MB（共享）
  - 内存: DDR5 4800MHz（XMP: 6000MHz）32GB（双通道）
  - 主板: 微星B760M Gaming Plus WiFi D5
- **软件环境**
  - OS: debian12 Bookworm desktop
  - 编译器: GCC 12.2.0
  - 线程库: OpenMP 4.0+

3.2　性能测试结果

不同线程数性能对比（采样点数：100,000,000）：

线程数	计算时间(s)	π 估算值	加速比	并行效率
1	0.827981	3.1415846	1.00	100%
2	0.407905	3.1415845	2.03	101.5%
4	0.204186	3.1415845	4.06	101.3%
6	0.138680	3.1415845	5.97	99.5%
10	0.135634	3.1415845	6.10	61%
12	0.122845	3.1415845	6.74	56.2%
16	0.102336	3.1415845	8.09	50.6%

不同采样点数精度测试（线程数：6）

采样点数	计算时间(s)	π 估算值	绝对误差	相对误差
1,000	0.000174	3.1280000	1.25e-02	4.32e-03
10,000	0.000250	3.1372000	4.39e-03	1.40e-03
100,000	0.001318	3.1418800	2.87e-04	9.15e-05
1,000,000	0.002677	3.1415840	8.65e-06	2.75e-06

采样点数	计算时间(s)	π 估算值	绝对误差	相对误差
10,000,000	0.015584	3.1415892	3.45e-06	1.10e-06
100,000,000	0.138680	3.1415845	8.15e-06	2.60e-06

3.3 加速比分析

线性加速特性：

- 低线程数下接近理想加速比
- 随着线程数增加，缓存竞争和调度开销显现
- 16 线程仍保持 8.09 倍加速，展现出良好的可扩展性

精度收敛性：

- 蒙特卡洛方法误差与 $\sqrt{N}$ 成反比
- 随机性导致不同运行结果略有差异

4 结论与优化措施

1. 并行效果显著：OpenMP 实现了良好的并行加速，最高加速比达 8.09 倍
2. 随机数优化关键： `rand_r()` 替代 `rand()` 是性能提升的关键因素
3. 算法优化有效：避免开平方运算进一步提升计算效率
4. 扩展性良好：在测试范围内保持较高的并行效率

通过本实验深入理解了 OpenMP 并行编程模型和蒙特卡洛方法的实现细节。随机数生成的优化体现了并行计算中线程安全和性能平衡的重要性，为今后的并行算法设计提供了宝贵经验。