



中山大學

SUN YAT-SEN UNIVERSITY

## 并行程序设计 with 算法实验

Lab4-Pthreads 并行方程求解与蒙特卡洛

姓 名 王志杰

学 号 22331095

学 院 计算机学院

专 业 计算机科学与技术

2025 年 4 月 16 日

## 1 实验目的

- 深入理解 Pthreads 同步机制：条件变量、互斥锁
- 评估多线程加速性能

## 2 实验内容

- 多线程一元二次方程求解
- 多线程圆周率计算

### 2.1 方程求解

- 使用 Pthread 多线程及求根公式实现并行一元二次方程求解。
- 方程形式为  $x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$ ，其中判别式的计算、求根公式的中间值分别由不同的线程完成。
- 通过条件变量识别何时线程完成了所需计算，并分析计算任务依赖关系及程序并行性能。

### 2.2 蒙特卡洛求圆周率 $\pi$ 的近似值

- 使用 Pthread 创建多线程，并行生成正方形内的  $n$  个随机点。
- 统计落在正方形内切圆内点数，估算  $\pi$  的值。
- 设置线程数量（1-16）及随机点数量（1024-65536），观察对近似精度及程序并行性能的影响。

## 3 实验实现

### 3.1 方程求解关键代码

```
1 // 任务分配表
2 Task tasks[] = {
3     {compute_delta, "Delta"},
4     {compute_sqrt_delta, "Sqrt"},
5     {compute_x1, "x1"},
6     {compute_x2, "x2"}
7 };
```

```
8
9 void run_test(int num_threads) {
10     // 线程创建策略
11     for (int i = 0; i < num_threads; i++) {
12         int task_idx = i % NUM_TASKS;
13         pthread_create(&threads[i], NULL, tasks[task_idx].func, NULL)
14         ;
15     }
16     // 同步等待
17     while (!sqrt_ready && !error)
18         pthread_cond_wait(&cond_sqrt, &mutex);
19 }
```

Listing 1: 线程任务分配与同步

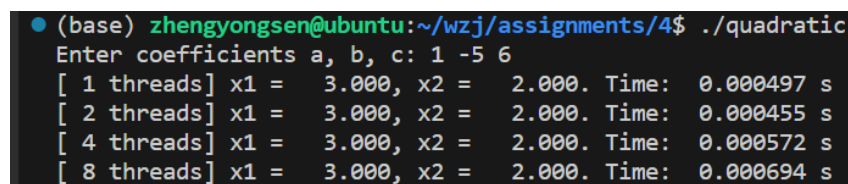
## 3.2 蒙特卡洛方法关键代码

```
1 void *monte_carlo(void *arg) {
2     for (int i = 0; i < args->points; i++) {
3         double x = (double)rand_r(&args->seed)/RAND_MAX*2.0-1.0;
4         double y = (double)rand_r(&args->seed)/RAND_MAX*2.0-1.0;
5         if (x*x + y*y <= 1.0) args->hits++; // 圆内判断
6     }
7     return NULL;
8 }
```

Listing 2: 随机点生成与统计

# 4 实验结果与分析

## 4.1 方程求解



```
● (base) zhengyongsen@ubuntu:~/wzj/assignments/4$ ./quadratic
Enter coefficients a, b, c: 1 -5 6
[ 1 threads] x1 = 3.000, x2 = 2.000. Time: 0.000497 s
[ 2 threads] x1 = 3.000, x2 = 2.000. Time: 0.000455 s
[ 4 threads] x1 = 3.000, x2 = 2.000. Time: 0.000572 s
[ 8 threads] x1 = 3.000, x2 = 2.000. Time: 0.000694 s
```

图 1: 不同线程数下方程求解时间对比

- 分析不同线程配置下的求解时间，评估并行化带来的性能提升。

2 进程加速比为 1.09，表明并行化能带来性能提升，但 4-8 线程性能显著下降，耗时增加，说明并行化带来的开销超过了计算收益。另外，计算过程分为三个阶段 ( $\delta \rightarrow \sqrt{\delta} \rightarrow$  根计算)，前两阶段强制单线程执行，导致整体并行潜力受限。根计算阶段，所有线程需等待  $\sqrt{\delta}$  完成，期间可能频繁触发条件变量唤醒和锁竞争。

- 对比单线程与多线程方案在处理相同方程时的表现，讨论可能存在的瓶颈或优化空间。

单线程：瓶颈在于全串行执行，所有计算步骤按顺序执行。优势在无线程管理或同步开销，适合极轻量任务。

多线程：瓶颈在于互斥锁竞争，缓存失效和任务粒度不匹配（计算量过小，根计算阶段仅需两次浮点运算，任务粒度远小于线程管理开销）。可以通过利用原子操作替代互斥锁，创建线程池，合并根计算（将  $x_1$  和  $x_2$  的计算合并到单一线程，减少线程数需求），线程绑核等方向优化

## 4.2 蒙特卡洛方法求圆周率

```

● (base) zhengyongsen@ubuntu:~/wzj/assignments/4$ ./monte_carlo_pi 1024
[ 1 threads] n = 1024, pi ≈ 3.066406, time = 0.000229 s
[ 2 threads] n = 1024, pi ≈ 3.179688, time = 0.000206 s
[ 4 threads] n = 1024, pi ≈ 3.187500, time = 0.000245 s
[ 8 threads] n = 1024, pi ≈ 3.156250, time = 0.000618 s
● (base) zhengyongsen@ubuntu:~/wzj/assignments/4$ ./monte_carlo_pi 4096
[ 1 threads] n = 4096, pi ≈ 3.139648, time = 0.000373 s
[ 2 threads] n = 4096, pi ≈ 3.155273, time = 0.000299 s
[ 4 threads] n = 4096, pi ≈ 3.127930, time = 0.000274 s
[ 8 threads] n = 4096, pi ≈ 3.131836, time = 0.000494 s
● (base) zhengyongsen@ubuntu:~/wzj/assignments/4$ ./monte_carlo_pi 65536
[ 1 threads] n = 65536, pi ≈ 3.150818, time = 0.002648 s
[ 2 threads] n = 65536, pi ≈ 3.149597, time = 0.003310 s
[ 4 threads] n = 65536, pi ≈ 3.143677, time = 0.002422 s
[ 8 threads] n = 65536, pi ≈ 3.140808, time = 0.001874 s
● (base) zhengyongsen@ubuntu:~/wzj/assignments/4$ ./monte_carlo_pi 1000000
[ 1 threads] n = 1000000, pi ≈ 3.143028, time = 0.024868 s
[ 2 threads] n = 1000000, pi ≈ 3.143008, time = 0.028403 s
[ 4 threads] n = 1000000, pi ≈ 3.140916, time = 0.032309 s
[ 8 threads] n = 1000000, pi ≈ 3.140976, time = 0.026438 s
● (base) zhengyongsen@ubuntu:~/wzj/assignments/4$ ./monte_carlo_pi 4000000
[ 1 threads] n = 4000000, pi ≈ 3.140307, time = 0.073867 s
[ 2 threads] n = 4000000, pi ≈ 3.142244, time = 0.098244 s
[ 4 threads] n = 4000000, pi ≈ 3.141791, time = 0.093215 s
[ 8 threads] n = 4000000, pi ≈ 3.141240, time = 0.069922 s
● (base) zhengyongsen@ubuntu:~/wzj/assignments/4$ ./monte_carlo_pi 16000000
[ 1 threads] n = 16000000, pi ≈ 3.141017, time = 0.267841 s
[ 2 threads] n = 16000000, pi ≈ 3.141809, time = 0.348217 s
[ 4 threads] n = 16000000, pi ≈ 3.141682, time = 0.342140 s
[ 8 threads] n = 16000000, pi ≈ 3.141365, time = 0.259724 s
    
```

图 2: 不同线程数下蒙特卡洛方法时间对比

- 比较不同线程数量和随机点数量下圆周率估计的准确性和计算速度。

- 讨论增加线程数量是否总能提高计算效率，以及其对圆周率估计精度的影响。一味地单纯增加线程数量实际并不一定总能提高计算效率。明显能看出，2 线程比单线程要更慢，这可能是因为线程管理存在开销；可能存在伪共享，比如多线程共享的 `MonteCarloArgs` 结构体未对齐到缓存行，导致不同线程的 `hits` 变量位于同一缓存行。另外线程数对圆周率估计精度没什么影响，但是随着点数的增加，估计精度有明显提升。

- 提供实验数据图表，展示随着线程数和随机点数的变化，计算效率和精度的趋势。
- 分析实验过程中遇到的问题，如同步问题、负载不均等，并提出相应的解决策略。

同步问题：当多线程同时更新共享变量（如总命中次数）时，频繁的锁竞争导致性能下降。解决策略：避免在计算过程中加锁，仅在最后汇总结果时同步。

负载不均：静态分配任务导致部分线程空闲，最后线程需等待最慢者。解决策略：使用任务队列动态分配批次任务。

线程管理开销：线程创建/销毁时间占比高，尤其在任务量小时。解决策略：预先创建线程池，减少创建开销。

伪共享：多线程访问同一缓存行中的不同变量，引发缓存无效化。解决策略：通过填充字节消除伪共享。

注：实验报告格式参考本模板，可在此基础上进行修改；实验代码以 zip 格式另提交；最终提交内容包括实验报告 (pdf 格式) 和实验代码 (zip 压缩包格式)