

并行程序设计 with 算法实验 4

实验	Pthread实现蒙特卡洛求 π	专业	计算机科学与技术
学号	21311525	姓名	马梓培
Email	mazp@mail2.sysu.edu.cn	完成日期	2024/04/15

1. 实验要求

- 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 π 的值
- 设置线程数量（1-16）及随机点数量（1024-65536）
- 分析近似精度及程序并行性能

2. 蒙特卡洛近似求 π 原理

借鉴自[知乎](#)。

蒙特卡洛模拟是最简单的随机抽样方式。随机抽样是一种严重依赖随机性来解决原则上确定性问题的计算算法。蒙特卡罗采样的艺术在于我们可以用它来解决一些其他方法难以解决的数学问题。

模拟的工作原理是在 $r = 1$ 的平方中生成随机点。同时这个 r 是正方形内四分之一的想象圆的半径。根据点到坐标系中心的距离，可以判断该点是在假想的四分之一圆内还是在假想的四分之一圆外。这是蒙特卡罗模拟的切入点。蒙特卡罗模拟将“射击”正方形中的点，同时再想象这些点“射击”在四分之一圆的内部和外部。通过四分之一圆内外的点的比例可以确定 π 。

更正式地说：

四分之一的圆面积为 $S1 = \frac{r^2}{4}$ ，正方形的面积为 $S2=r^2$

k和m分别是通过蒙特卡罗模拟，落在四分之一的圆面积S1和正方形的面积为S2的点数。

$$\frac{S1}{S2} = r^2\pi/4/r^2 = \pi = k/m$$

$$\pi = \frac{4k}{m}$$

3. Pthread蒙特卡洛近似求 π

首先，创建ThreadData结构体，方便在fork的线程中使用：

```

1 struct ThreadData {
2     int thread_id;
3     long long num_points;        // 每个线程生成的点数
4     long long count_in_circle;   // 落在圆内的点数
5 };

```

其次，根据thread_id作为seed，随机生成不同的二维点。考虑到srand()函数生成的是[0, 1]的均匀采样，我将其-0.5，得到[-0.5, 0.5]的均匀采样。因此，圆的半径被设置为0.5。即，若生成的点到原点的距离 > 0.5，ThreadData.count_in_circle++。

```

1 ThreadData* data = (ThreadData*)arg;
2 double x, y;
3 double radius = 0.5;
4
5 srand(time(NULL) ^ (unsigned int)data->thread_id);
6
7 for (int i = 0; i < data->num_points; i++) {
8     x = (double)rand() / RAND_MAX; // 生成0到1之间的随机数
9     y = (double)rand() / RAND_MAX;
10    if (sqrt((x - 0.5) * (x - 0.5) + (y - 0.5) * (y - 0.5)) <= radius) {
11        data->count_in_circle++;
12    }
13 }

```

常规的创建线程，合并线程：

```

1 for (int i = 0; i < thread_cnt; i++) {
2     data[i].thread_id = i; // 分配线程ID
3     data[i].num_points = points_per_thread;
4     pthread_create(&threads[i], NULL, generate_points, (void*)&data[i]);
5 }
6
7 long long total_in_circle = 0;
8 for (int i = 0; i < thread_cnt; i++) {
9     pthread_join(threads[i], NULL);
10    total_in_circle += data[i].count_in_circle;
11 }

```

最后，根据公式，计算 π ：

```

1 double pi_estimate = 4.0 * total_in_circle / total_points;

```

同样的，计算时间也与上次作业相同：

```

1 // Barrier
2 pthread_mutex_lock(&mutex1);
3 counter++;
4 if (counter == thread_cnt) {
5     counter = 0;
6     pthread_cond_broadcast(&cond_var);
7 }

```

```
8  else {
9      while (pthread_cond_wait(&cond_var, &mutex1) != 0);
10 }
11 pthread_mutex_unlock(&mutex1);
12 clock_gettime(CLOCK_MONOTONIC, &start_time);
13 ...
14 clock_gettime(CLOCK_MONOTONIC, &end_time);
15 elapsed_time = (end_time.tv_sec - start_time.tv_sec);
```

4. 实验结果

N = 1024

```
1  > ./pi.out 1
2  Cost time: 0.000029
3  Estimated Pi: 3.199219
4  > ./pi.out 2
5  Cost time: 0.000047
6  Estimated Pi: 3.101562
7  > ./pi.out 4
8  Cost time: 0.000022
9  Estimated Pi: 3.058594
10 > ./pi.out 8
11 Cost time: 0.000012
12 Estimated Pi: 3.082031
13 > ./pi.out 16
14 Cost time: 0.000003
15 Estimated Pi: 3.179688
```

N = 2048

```
1  > ./pi.out 1
2  Cost time: 0.000057
3  Estimated Pi: 3.060547
4  > ./pi.out 2
5  Cost time: 0.000061
6  Estimated Pi: 3.162109
7  > ./pi.out 4
8  Cost time: 0.000016
9  Estimated Pi: 3.199219
10 > ./pi.out 8
11 Cost time: 0.000046
12 Estimated Pi: 3.070312
13 > ./pi.out 16
14 Cost time: 0.000012
15 Estimated Pi: 3.156250
```

N = 4096

```
1  > ./pi.out 1
```

```
2 Cost time: 0.000187
3 Estimated Pi: 3.125000
4 > ./pi.out 2
5 Cost time: 0.000120
6 Estimated Pi: 3.152344
7 > ./pi.out 4
8 Cost time: 0.000037
9 Estimated Pi: 3.118164
10 > ./pi.out 8
11 Cost time: 0.000095
12 Estimated Pi: 3.152344
13 > ./pi.out 16
14 Cost time: 0.000014
15 Estimated Pi: 3.159180
```

N = 8192

```
1 > ./pi.out 1
2 Cost time: 0.000301
3 Estimated Pi: 3.125488
4 > ./pi.out 2
5 Cost time: 0.000120
6 Estimated Pi: 3.197754
7 > ./pi.out 4
8 Cost time: 0.000123
9 Estimated Pi: 3.118164
10 > ./pi.out 8
11 Cost time: 0.000114
12 Estimated Pi: 3.106445
13 > ./pi.out 16
14 Cost time: 0.000067
15 Estimated Pi: 3.079590
```

N = 16384

```
1 > ./pi.out 1
2 Cost time: 0.000497
3 Estimated Pi: 3.144531
4 > ./pi.out 2
5 Cost time: 0.000499
6 Estimated Pi: 3.129150
7 > ./pi.out 4
8 Cost time: 0.000301
9 Estimated Pi: 3.131104
10 > ./pi.out 8
11 Cost time: 0.000056
12 Estimated Pi: 3.138672
13 > ./pi.out 16
14 Cost time: 0.000142
15 Estimated Pi: 3.193604
```

N = 32768

```
1  > ./pi.out 1
2  Cost time: 0.001188
3  Estimated Pi: 3.155396
4  > ./pi.out 2
5  Cost time: 0.000951
6  Estimated Pi: 3.144287
7  > ./pi.out 4
8  Cost time: 0.000286
9  Estimated Pi: 3.125977
10 > ./pi.out 8
11 Cost time: 0.000456
12 Estimated Pi: 3.140747
13 > ./pi.out 16
14 Cost time: 0.000082
15 Estimated Pi: 3.131226
```

N = 65536

```
1  > ./pi.out 1
2  Cost time: 0.002153
3  Estimated Pi: 3.156189
4  > ./pi.out 2
5  Cost time: 0.000963
6  Estimated Pi: 3.127136
7  > ./pi.out 4
8  Cost time: 0.000812
9  Estimated Pi: 3.141846
10 > ./pi.out 8
11 Cost time: 0.000563
12 Estimated Pi: 3.142517
13 > ./pi.out 16
14 Cost time: 0.000378
15 Estimated Pi: 3.131592
```