实验 1

学号 PB15000027
姓名 韦清

实验题目:
利用 MPI，OpenMP 编写简单的程序，测试并行计算系统性能

实验环境(操作系统,编译器,硬件配置等):
Ubuntu 16.04 (Win10 WSL)
Intel® Core™ i7-8650U CPU @ 1.90GHz 2.11GHz
编译器:
gcc (Ubuntu 5.4.0-6ubuntu1~16.04.9) 5.4.0 20160609

算法设计与分析(写出解题思路和实现步骤)
1. 求素数个数
定义一个 isPrime 函数，如果参数是素数，返回 1，否则返回 0.main 函数中，遍历 2 到 n 的所有数，count+=isPrime(i)，即为结果。
isPrime 函数中，对于每一个输入 x，遍历 2 到 sqrt(x)，如果 x 能被之间的数整除，则返回 0，如果遍历完成没有发现可以整除 x 的数，返回 1.
2. 求 Pi 值
可以证明，$\pi = \int_0^1 \frac{4}{1+x^2} dx$
故该问题转化为一个求微积分的问题。将 0 到 1 分为 n 份，将每份看成一个矩形，求面积，再对其求和即可。

核心代码(写出算法实现的关键部分,如核心的循环等)
1. **求素数个数**
OpenMP

```
omp_set_num_threads(threadnum);
#pragma omp parallel for reduction(+:count)
for(int j=2;j<=n;j++) {
    count+=isPrime(j);
}
```

MPI

```
for(int j=myid;j<=n;j+=size) {
    count += isPrime(j);
}
MPI_Reduce(&count, &sum, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);
```

其中 myid 为当前进程在所有进程中的 rank，size 为进程数
2. **求 pi 值**
OpenMP

```
sum=0.0;
#pragma omp parallel for reduction(+:sum)
for(int j=0;j<n;j++) {
    double x = h*(j+0.5);
    sum += 4.0/(1.0+x*x);
}
pi = h*sum;
```

其中 **h=1.0/n**

MPI

```
h = 1.0/n;
sum=0.0;
int j;
for(j=myid;j<n;j+=size) {
    x = h*(j+0.5);
    sum += 4.0/(1+x*x);
}
sum = sum*h;
MPI_Reduce(&sum, &pi, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);
```

实验结果:

实验结果按如下格式排列

1. **求素数个数**

OpenMP

运行时间

规模\线程数	1	2	4	8
1000	0.000000ms	**0.000000ms**	**0.000000ms**	**0.000000ms**
10000	**1.562500ms**	**0.000000ms**	**0.000000ms**	**0.000000ms**
100000	**7.812500ms**	9.375000ms	**12.500000ms**	**25.000000ms**
500000	**76.562500ms**	**98.437500ms**	**121.87500ms**	**189.06250ms**

加速比

规模\线程数	1	2	4	8
1000	**1**	**Nan**	**Nan**	**Nan**
10000	**1**	**Nan**	**Nan**	**Nan**
100000	**1**	**0.833333**	**0.625**	**0.3125**
500000	**1**	**0.777778**	**0.628205**	**0.404959**

MPI

时间单位均为 **ms**

运行时间

规模\线程数	1	2	4	8
1000	**0.1302242279**	**0.0791311264**	**0.1543045044**	**1.2607574463**

10000	0.408395004	0.328682831	0.1437425613	0.257253647
100000	7.1611404419	6.976151466	4.9578666687	3.215874786
500000	63.9662275314	64.9895429611	41.487789141	40.5387163162

加速比

规模\线程数	1	2	4	8
1000	1	1.645676	0.843943	0.10329
10000	1	1.24252	2.841156	1.587519
100000	1	1.026517	1.4444	2.226809
500000	1	0.984254	1.541809	1.577905

2. 求 pi 值

OpenMP

运行时间

规模\线程数	1	2	4	8
1000	0.015625ms	0.031250ms	0.062500ms	0.125000ms
10000	0.140625ms	0.156250ms	0.125000ms	0.468750ms
50000	0.703125ms	0.718750ms	0.734375ms	1.093750ms
100000	1.406250ms	1.468750ms	1.437500ms	2.390625ms

加速比

规模\线程数	1	2	4	8
1000	1	0.5	0.25	0.125
10000	1	0.9	1.125	0.3
50000	1	0.978261	0.957447	0.642857
100000	1	0.957447	0.978261	0.588235

MPI

时间单位为 ms

运行时间

规模\线程数	1	2	4	8
1000	0.0174069405	0.0091414452	0.0069139004	0.0151183605
10000	0.1520159245	0.0737934113	0.0383965969	0.0270650387

50000	0.7405745983	0.3728165627	0.1844134331	0.1001121998
100000	1.4419374466	0.7505595684	0.3624238968	0.2081141472

加速比

规模\线程数	1	2	4	8
1000	1	1.9041781818	2.51767302	1.1513775254
10000	1	2.0600202894	3.9590988987	5.6166897149
50000	1	1.9864326242	4.015840769	7.397449906
100000	1	1.9211499091	3.9785937388	6.928589267

分析与总结

在使用 OpenMP 时，出现了进程越多，时间越长的情况，可能原因如下：

1. 程序运行时间过短，故统计的时间不精确。这种原因主要可能发生于求素数程序。在求 pi 程序中，由于程序运行时间短，所以将每个程序都运行 1000 次，统计总时间，再除以 1000，来保证结果的准确性；在求素数的程序中，由于规模较大时耗时长，所以此处仅让其循环 10 次，求平均时间。
2. 程序计算所需时间端，MPI 的通信和 OpenMP 的通信花费了较大时间
3. 每个线程/进程负载不均衡，导致出现了瓶颈。
4. 编译器进行了一些优化，使得某些情况下执行时间变短。