lab1

韩佳乐 PB16051152

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
Ubuntu 16.04	gcc mpicc	双核 4G内存

算法设计与分析

求素数个数

检测一个整数a是否为素数：如果从2到$\sqrt{a}$均无法整除a，那么a即为素数。由于除2以外的所有偶数都不是素数，所以只需要检测1~n中所有的奇数(这里把1当作素数，把2当作非素数，虽然有违常理，但对求素数个数没有影响)。

求 Pi 值

$$\pi = 4\arctan 1 = 4\int_0^1 \frac{dx}{1+x^2} = 4\int_0^1 \sum_{n=0}^{\infty} (-x^2)^n dx$$

$$= 4\sum_{n=0}^{\infty} \int_0^1 (-x^2)^n dx = 4\sum_{n=0}^{\infty} \frac{(-1)^n}{2n+1}.$$

核心代码

求素数个数 MPI

初始化MPI环境，获取并行环境参数(总线程数、本地进程编号等)。

```
MPI_Init(&argc, &argv);
MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
MPI_Comm_rank(MPI_COMM_WORLD, &myid);
```

从0号进程获取输入n，并将其广播出去。

```
if(myid == 0){
    printf("输入n:");
    scanf("%d",&n);
    startwtime = MPI_Wtime();
}
MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);   //将n值广播出去
```

并行计算，然后将结果进行规约。pi即为最终结果

```
sum = 0;
for(int i = myid*2+1; i <= n; i += numprocs*2){
    sum += isPrime(i);
}
mypi = sum;
MPI_Reduce(&mypi, &pi, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);  //规约
```

求素数个数 OpenMP

openMP相对比较简单，omp_set_num_threads()设置线程数，#pragma omp parallel for语句开启并行化，reduction(+:sum)表示各线程的运行结果sum最终要进行加法运算。

```
omp_set_num_threads(2);
#pragma omp parallel for reduction(+:sum)
for(int i = 1; i <= n; i += 2){
    sum += isPrime(i);
}
```

求 Pi 值 MPI

初始化MPI环境，获取并行环境参数(总线程数、本地进程编号等)。

```
MPI_Init(&argc, &argv);
MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
MPI_Comm_rank(MPI_COMM_WORLD, &myid);
```

从0号进程获取输入n，并将其广播出去。

```
if(myid == 0){
    printf("输入n:");
    scanf("%d",&n);
    startwtime = MPI_Wtime();
}
MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);   //将n值广播出去
```

并行计算，然后将结果进行规约。pi即为最终结果

```
sum = 0;
for(int i = myid; i <= n; i += numprocs){
    sum += (i % 2 == 0 ? 4.0 : (-4.0)) / (2 * i + 1);
}
mypi = sum;
MPI_Reduce(&mypi, &pi, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);  //规约
```

求 Pi 值 OpenMP

openMP相对比较简单，omp_set_num_threads()设置线程数，#pragma omp parallel for语句开启并行化，reduction(+:sum)表示各线程的运行结果sum最终要进行加法运算。

```
omp_set_num_threads(2);
#pragma omp parallel for reduction(+:sum)
for(int i = 0; i <= n; i++){
    sum += (i % 2 == 0 ? 4.0 : (-4.0)) / (2 * i + 1);
}
```

实验结果

求素数个数 MPI 运行时间(s)

规模/线程数	1	2	4	8
1000	0.000075	0.020078	0.035705	0.063428
10000	0.000652	0.017054	0.038764	0.070721

100000	0.029157	0.039937	0.057087	0.071655
1000000	0.404605	0.334772	0.280256	0.238929
5000000	3.335149	2.538442	2.142986	2.016233

求素数个数 MPI 加速比

规模/线程数	1	2	4	8
1000	1	0.004	0.002	0.001
10000	1	0.038	0.017	0.009
100000	1	0.730	0.511	0.407
1000000	1	1.209	1.444	1.693
5000000	1	1.314	1.556	1.654

求PI MPI 运行时间(s)

规模/线程数	1	2	4	8
10000	0.000087	0.004757	0.025594	0.040260
100000	0.000416	0.015015	0.036435	0.054088
1000000	0.003892	0.029813	0.045457	0.060422
10000000	0.047871	0.037937	0.055264	0.070782
100000000	0.410810	0.330348	0.282296	0.307018
1000000000	8.801296	5.527966	3.751501	2.940126

求PI MPI 加速比

规模/线程数	1	2	4	8
10000	1	0.018	0.003	0.002
100000	1	0.028	0.011	0.008
1000000	1	0.131	0.086	0.064
10000000	1	1.262	0.866	0.676
100000000	1	1.244	1.455	1.338

1000000000	1	1.592	2.346	2.994

分析与总结

1. 当数据比较小时，进程的创建和同步等因多进程而导致的开销是加速比小于1的主要原因。随着数据的增大，计算的开销越来越大，这种额外开销占比越来越小，多线程并行计算的优点逐渐体现出来，因此加速比也随之增加并且超过1。
2. 当数据较大时，随着线程数的增加，加速比虽然也在增加，但是增加速率很小，部分原因是多线程导致的额外开销随着线程数增加而增加，更重要的原因在于当线程数超过CPU核数时(本实验为双核)CPU占有率会达到极限，很难再有提升。