实验一

姓名：王嵘晟
学号： PB17111614

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： Windows 10
IDE： Visual Studio 2019 X64 Debug 模式 MPI环境： MS-MPI V10
硬件配置： Intel CORE i7 6550U
OPENMP环境： Visual Studio 2019 自带

算法设计与分析

1. MPI 求素数个数： 由于素数除了2以外都是奇数，所以从1开始每次加2来判断该奇数是否是素数，若是素数则计数++，否则继续向下直到遍历到n为止。具体实现为并行部分：对每个线程均计算当前循环 i 的值是否为素数，只有 mypid=0 输出结果。串行：只用线程0来遍历，找素数个数。
2. OpenMP 求素数个数：由于素数除了2以外都是奇数，所以从1开始每次加2来判断该奇数是否是素数，若是素数则计数++，否则继续向下直到遍历到n为止。具体实现为并行部分：parallel for 对计数 sum 进行共享，并做求和操作。串行：单循环遍历，找素数个数。
3. MPI 计算π: 利用公式：

$$\int_0^1 \frac{4}{1+x^2} = \pi$$

将区间[0, 1]作 n 等分，即

$$h = \frac{1}{n}$$

对于每个长为h的小区间，计算

$$\frac{4}{1+(\frac{i}{n})^2}$$

的值后作累加即可得到π。并行部分：对于每个线程，计算

$$\frac{4}{1+(\frac{i}{n})^2}$$

并累加，通过通信传递给最终结果。串行：单循环累加。
4. OpenMP 计算π: 利用公式：

$$\int_0^1 \frac{4}{1+x^2} = \pi$$

将区间[0, 1]作 n 等分，即

$$h = \frac{1}{n}$$

对于每个长为h的小区间，计算

$$\frac{4}{1+(\frac{i}{n})^2}$$

的值后作累加即可得到π。并行部分：parallel for 对 sum 进行共享，计算

$$\frac{4}{1+(\frac{i}{n})^2}$$

并累加。串行：单循环累加。

核心代码

1. MPI 求素数个数：

```
// MPI 初始化
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &mypid);
MPI_Comm_size(MPI_COMM_WORLD, &mpi_threads_num);

// 信息传递
MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);
startTime1 = MPI_Wtime();

// 计算质数数量并存储在本地local
for (int i = mypid * 2 + 1; i <= n; i += mpi_threads_num * 2)
{
    local += isPrime(i);
}

// 归约，将各个local中数据集中到sum中
MPI_Reduce(&local, &sum, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);
endTime1 = MPI_Wtime();
t1 = endTime1 - startTime1;
```

2. OpenMP 求素数个数：

```
startTime1 = clock();
// parallel for 并行部分
#pragma omp parallel for reduction(+:sum)
    for (i = 1; i <= n; i+=2)
    {
        sum += isPrime(i);
    }
    endTime1 = clock();
```

```
    delta1 = endTime1 - startTime1;
    printf("OpenMp:\n素数个数:\t%d\n时间:%ld ms\n\n", sum, delta1);
// 串行部分
    sum = 0;
    startTime2 = clock();
    for (i = 1; i <= n; i+=2)
    {
        sum += isPrime(i);
    }
    endTime2 = clock();
    delta2 = endTime2 - startTime2;
    printf("串行:\n素数个数:\t%d\n时间:%ld ms\n", sum, delta2);
```

3. MPI 计算π:

```
// MPI 初始化
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &mypid);
MPI_Comm_size(MPI_COMM_WORLD, &mpi_threads_num);

// 信息传递
MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);
startTime1 = MPI_Wtime();

// 计算质数数量并存储在本地local
for (i = mypid + 1; i <= n; i += mpi_threads_num)
{
    x = h * ((double)i - 0.5);
    local += fx(x);
}

// 归约，将各个local中数据集中到sum中
MPI_Reduce(&local, &pi, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);
pi *= h;
endTime1 = MPI_Wtime();
t1 = endTime1 - startTime1;

if (mypid == 0)
{
    printf("Number of threads: %d \n", mpi_threads_num);
    printf("Time:\t%f s\n", t1);
    printf("Pi:\t%.12lf\n\n", pi);
}

// 计算单线程时间和加速比
pi = 0;
startTime2 = MPI_Wtime();
if (mypid == 0)
{
    for (int i = 1; i <= n; i ++)
    {
        x = h * ((double)i - 0.5);
```

```
            pi += fx(x);
        }
        pi *= h;
        endTime2 = MPI_Wtime();
        t2 = endTime2 - startTime2;
        printf("Serial Computing in thread %d\n", mypid);
        printf("Time:\t%f s\n", t2);
        printf("Pi:\t%.12lf\n\n", pi);

        printf("Speedup: %f", t2 / t1);
    }
    MPI_Finalize();
```

4. OpenMP 计算π:

```
startTime1 = clock();
#pragma omp parallel for reduction(+:sum)
    for (i = 1; i <= n; i++)
    {
        x = h * ((double)i - 0.5);
        sum += fx(x);
    }
    sum *= h;
    endTime1 = clock();
    delta1 = endTime1 - startTime1;
    printf("OpenMp:\npi:\t%.12lf\n时间:%ld ms\n\n", sum, delta1);

    sum = 0.0;
    startTime2 = clock();
    for (i = 1; i <= n; i++)
    {
        x = h * ((double)i - 0.5);
        sum += fx(x);
    }
    sum *= h;
    endTime2 = clock();
    delta2 = endTime2 - startTime2;
    printf("串行:\npi:\t%.12lf\n时间:%ld ms\n\n", sum, delta2);
    if (delta1 != 0)
        printf("加速比:\t %.6lf", (double)delta2 / (double)delta1);
```

实验结果

1. MPI 求素数个数:

时间

规模\进程数	1	2	4	8
1000	0.000079s	0.000608s	0.001082s	0.002128s

规模\进程数	1	2	4	8
10000	0.001060s	0.000902s	0.001327s	0.002282s
100000	0.034363s	0.023632s	0.021144s	0.019068s
500000	0.271856s	0.159187s	0.138709s	0.159300s

加速比(与串行相比)

规模\进程数	1	2	4	8
1000	1	0.087142	0.057785	0.034591
10000	1	1.722548	1.215524	0.633528
100000	1	1.232086	1.184049	1.477082
500000	1	1.549657	2.016242	1.775610

2. OpenMP 求素数个数：

时间（此处使用毫秒计时）

规模\进程数	1	2	4	8
1000	0 ms	1 ms	4 ms	7 ms
10000	2 ms	3 ms	5 ms	6 ms
100000	47 ms	29 ms	30 ms	26 ms
500000	462 ms	215 ms	151 ms	132 ms

加速比(与串行相比)

规模\进程数	1	2	4	8
1000	1	0.305882	0.123809	0.073239
10000	1	1.333333	0.600000	0.333333
100000	1	1.724138	1.800000	2.115385
500000	1	2.237209	3.675497	4.030303

3. MPI 计算π:

时间

规模\进程数	1	2	4	8
1000	0.000114s	0.000840s	0.001210s	0.002269s
10000	0.000881s	0.001445s	0.001232s	0.002322s
50000	0.004503s	0.003201s	0.002679s	0.003396s

规模\进程数	1	2	4	8
100000	0.004559s	0.003587s	0.002715s	0.003866s

加速比(与串行相比)

规模\进程数	1	2	4	8
1000	1	0.101822	0.091285	0.044335
10000	1	0.599515	0.824890	0.392394
50000	1	1.484241	1.663506	1.300571
100000	1	1.238304	1.663781	1.137211

4. OpenMP 计算π:

时间（此处使用毫秒计时）

规模\进程数	1	2	4	8
1000	1 ms	1 ms	2 ms	6 ms
10000	1 ms	3 ms	3 ms	7 ms
50000	6 ms	4 ms	7 ms	13 ms
100000	11 ms	10 ms	13 ms	16 ms

加速比(与串行相比)

规模\进程数	1	2	4	8
1000	1	0.331081	0.197581	0.075617
10000	1	0.333333	0.666667	0.285714
50000	1	1.500000	1.142857	0.538462
100000	1	1.200000	0.923007	0.812500

实验结论

1. 对于求素数个数，OpenMP 的加速性能要好于 MPI。但当规模较小时由于计算消耗资源较少，并行计算带来的通信时间使得并行处理的时间要比串行还长。
2. 对于计算π， MPI 的加速性能要好于 OpenMP。但当规模较小时由于计算消耗资源较少，并行计算带来的通信时间使得并行处理的时间要比串行还长。
3. 综合来看：计算π所消耗的计算资源比求素数个数要小，所以使用并行计算来计算π效果普遍不理想。这个问题在增大n后会得到一定程度的解决。
4. 当处理计算消耗较大的程序时，应该选择 OpenMP 做并行处理，反之使用 MPI。