实验一

学号：SA17011125
姓名：吴燕晶

一．求素数个数

（一）实验题目：求素数个数

① 实验描述：给定正整数 n,编写程序计算出所有小于等于 n 的素数的个数

② 实验要求：需要测定 n=1000;10000;100000;500000 时程序运行的时间

（二）实验环境

① 虚拟机：VMware

② 操作系统：Ubuntu16.04

③ 内存：4G

④ 处理器：4

（三）算法设计与分析

1．values 函数

① 设置线程数

② 定义一个全局变量，来统计小于等于 n 的素数的个数

③ 并行的对于从 2 到 n 的数判断每一个数是不是素数

④ 输出素数个数和所需的时间，以及加速比

2．isprime 函数：判断一个函数是不是素数

（四）核心代码

1．并行的对于从 2 到 n 的数判断每一个数是不是素数

（1）openMP

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

（2）MPI

```
if(myrank==0){
  printf("the n is %d\n", n);
  starttime = MPI_Wtime();
}
MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);
sum = 0;
for(i=myrank*2+1; i<=n; i+=nprocs*2){
  sum += isPrime(i);
}
mynumber = sum;
MPI_Reduce(&mynumber, &number, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);
if(myrank==0){
  printf("the result is %d\n", number);
  endtime = MPI_Wtime();
  time1 = endtime - starttime;
  printf("the parallel time = %lf\n", time1);
}
```

2．isprime 函数

```
// judge if n is prime or not
int isPrime(int n){
  int flag = 1;
  int middle = sqrt((double)n);
  int i;
  for(i=2; i<=middle; i++){
    if(n%i==0){
      flag = 0;
      break;
    }
  }
  return flag;
}
```

（五）实验结果

1．用 openMP 实现

运行时间

规模\线程数	1	2	4	8
1000	0.000035	0.000055	0.000180	0.000256
10000	0.000445	0.000241	0.000116	0.000248
100000	0.009336	0.006257	0.003265	0.003878
500000	0.096890	0.052676	0.032234	0.030936

加速比

规模\线程数	1	2	4	8
1000	1	0.436364	0.138889	0.093750
10000	1	1.834025	3.818966	1.770161
100000	1	1.609078	3.587749	2.554152
500000	1	1.776559	2.945927	3.176106

2．用 MPI 实现

运行时间

规模\进程数	1	2	4	8
1000	0.000031	0.000025	0.000021	0.031841
10000	0.000443	0.000312	0.000114	0.024068
100000	0.012142	0.005189	0.009555	0.027547
500000	0.094530	0.055693	0.034141	0.047133

加速比

规模\进程数	1	2	4	8
1000	1	0.952830	1.149425	0.000764
10000	1	1.837796	3.886792	0.018019
100000	1	2.290775	1.529183	0.341806

500000	**1**	**1.695135**	**3.203587**	**4.369407**

（六）分析与总结

1．从 openmp 的运行时间和加速比上来看，当规模是 1000 的时候，运行时间比串行更慢，并且线程数越多运行时间越慢，这是因为当并行处理的规模太小的时候，本身计算的时间耗费是很少，而此时通信的时间耗费又很大。这就导致随着线程数的增加加速比反而下降。当规模数超过 1000，运行时间都会比串行更少。但是对于规模为 10000 和规模为 100000 的数据，在线程数为 1 到 4 的时候运行时间会随着线程数的增加而减少，加速比特而线程数为 8 的时候运行时间反而比线程数为 4 的时候增加了，加速比也下降了。而对于规模为 500000 的数据而言，它的运行时间在线程数为 1 到 8 的时候一直会随着线程数的增加而减少，加速比也随着线程数的增加而增加。

2．从 mpi 的运行时间和加速比上来看，当规模是 1000 的时候， 运行时间基本上是没有提高的，并且当进程数为 8 的时候运行时间和串行相比严重增长。这是主要是由于规模数太小导致的。当规模大于 1000 时，在进程数为 2 到 4 的时候，可以看到运行时间减少，程序加速了。但是当规模是为 10000 和 100000，进程数为 8 的时候，运行时间却增加了，加速比明显下降。而当规模是 500000 是，运行时间会随着进程数的增加而减少，加速比也会增加。以上的原因都是因为规模数的影响，当规模数太小，进程之间的通信太多，时间就会增加。

3．刚开始的时候我计算素数个数的时候，考虑了偶数的情况，但是实际上，偶数一定不是素数（2 除外），运行的时间会比上面的结果更长，并且加速比也最多在 1。这是因为计算偶数的进程拖慢了时间，影响了并行化的效率。所以我后面修改了代码。让它只计算奇数的情况，并且因为 1 既不是质数也不是合数，而 2 是质数，所以通过计算从 1 开始的素数个数刚好可以使得素数个数和真正的结果相等。

4．综上所述，数据规模和并行度对于并行的效率有很大的影响。我们要想达到最好的并行效果，需要两者之间的折中。

二．求 pi 值

（一）实验题目：求 pi 值

1．实验描述：给定迭代次数 n，编写程序计算 PI 的值

2．实验要求：算法必须使用近似公式求解；需要测定 n=1000;n=10000;n=50000;n=100000（逗号仅为清洗考虑）时程序运行时间

（二）实验环境

① 虚拟机：VMware

② 操作系统：Ubuntu16.04

③ 内存：4G

④ 处理器：1

（三）算法设计与分析

1. values 函数：

① 定义步伐 w

② 初始化 pi

③ 并行执行 n 次，每次计算一个小方格的值

（四）核心代码

1. Openmp

```
w = 1.0/n;
pi = 0.0;
#pragma omp parallel for reduction(+:pi) private(local)
for(i=0; i<n; i++){
  local = (i+0.5)*w;
  pi += 4.0/(1.0+local*local);
}
```

2. Mpi

```
int myrank, nprocs;
MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
double starttime, endtime, time1, time2;
if(myrank==0){
  printf("the n is %d\n", n);
  starttime = MPI_Wtime();
}

for(i = myrank; i<n; i+=nprocs){
  temp = (i+0.5)*w;
  local += 4.0/(1.0+temp*temp);
}
MPI_Reduce(&local, &pi, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);
if(myrank==0){
  printf("the parallel pi is %lf\n", pi*w);
  endtime = MPI_Wtime();
  time1 = endtime - starttime;
  printf("the parallel time = %lf\n", time1);
}
```

（五）实验结果

1. 用 openMP 实现

运行时间

规模\线程数	1	2	4	8
1000	0.000010	0.000063	0.000438	0.000350
10000	0.000045	0.0000627	0.000014	0.000084
50000	0.000229	0.000111	0.000057	0.000296
100000	0.000507	0.000220	0.000112	0.000487

加速比

规模\线程数	1	2	4	8
1000	1	0.063492	0.011416	0.011429
10000	1	1.629630	3.142857	0.535714
50000	1	1.972973	4.087719	0.743243
100000	1	2.0818182	4.142857	0.956879

2. 用 MPI 实现

运行时间

规模\进程数	1	2	4	8
1000	0.000012	0.000016	0.000014	0.008191

10000	**0.000055**	**0.000024**	**0.000144**	**0.000070**
100000	**0.00239**	**0.000258**	**0.000118**	**0.000099**
500000	**0.000457**	**0.000242**	**0.000184**	**0.000129**

加速比

规模\进程数	**1**	**2**	**4**	**8**
1000	**1**	**0.279412**	**0.316667**	**0.000553**
10000	**1**	**1.858586**	**0.535655**	**0.627986**
100000	**1**	**1.154201**	**2.738866**	**2.209135**
500000	**1**	**1.856441**	**2.602597**	**3.463956**

（六）分析与总结

1．从 openmp 的运行时间和加速比可以看出，当规模为 1000 时，运行时间会随着线程数的增加而增加，这是由于规模是太小的原因。当规模大于 1000 时，在线程数为 2 到 4 之间时，运行时间比串行的时候减少了，并且时间会随着线程数的增加而减少，加速比会随着时间的增加而增加，而当线程数为 8 的时候，运行时间会比串行所需的时间还要长，加速比迅速下降。这是由于规模数太小了，而线程数太多，线程之间的通行耗费了大量的时间，而计算只占总时间的一小部分。

2．从 mpi 的运行时间和加速比可以看出，当规模数为 1000 和 10000 时，运行时间比串行还要长，并且基本上运行时间会随着进程数的增加而增加，加速比是不断减少的。而当规模为 100000 和 500000 时，并行之后的运行时间会比串行的时间少，并且基本上来说运行时间会随着进程数的增加而增加，也就是加速比会随着进程数的增加而增加。

5．综上所述，在一定的范围内下提高进程数，会使得运行时间减少，提高加速比。但是进程数过多的时候，由于进程之间通行的影响却会使得运行时间增加，加速比减低。