

电子科技大学
计算机科学与工程学院

标准实验报告

(实验) 课程名称 分布式并行计算

电子科技大学教务处制表

电子科技大学

实验报告

学生姓名：朱若愚 学号：2022150501027 指导教师：李可欣

实验地点：A2-412

实验时间：2025.3.29

一、实验室名称： 计算机学院实验中心

二、实验项目名称：埃拉托斯特尼素数筛选算法并行及性能优化

三、实验学时：4 学时

四、实验原理：

埃拉托斯特尼是一位古希腊数学家，他在寻找整数 N 以内的素数时，采用了一种与众不同的方法：先将 $2 \sim N$ 的各数写在纸上：

在 2 的上面画一个圆圈，然后划去 2 的其他倍数；第一个既未画圈又没有被划去的数是 3，将它画圈，再划去 3 的其他倍数；现在既未画圈又没有被划去的第一个数是 5，将它画圈，并划去 5 的其他倍数……依此类推，一直到所有小于或等于 N 的各数都画了圈或划去为止。这时，画了圈的以及未划去的那些数正好就是小于 N 的素数。

这里，我们把 N 取 120 来举例说明埃拉托斯特尼筛法思想：

- 首先将 2 到 120 写出
 - 在 2 上面画一个圆圈，然后划去 2 的其它倍数，这时划去的是除了 2 以外的其它偶数
 - 从 2 往后一个数一个数地去找，找到第一个没有被划去的数 3，将它画圈，再划去 3 的其它倍数（以斜线划去）
 - 再从 3 往后一个数一个数地去找，找到第一个没有被划去的数 5，将它画圈，再划去 5 的倍数（以交叉斜线划去）
 - 再往后继续找，可以找到 9、11、13、17、19、23、29、31、37、41、43、47…将它们分别画圈，并划去它们的倍数（可以看到，已经没有这样的数了）
- 这时，小于或者等于 120 的各数都画上了圈或者被划去，被画圈的就是素数了

五、实验目的：

1. 使用 MPI 编程实现埃拉托斯特尼筛法并行算法。
2. 对程序进行性能分析以及调优。

六、实验内容：

优化 1：去掉偶数

优化思想利用“大于 2 的质数都是奇数”这一知识，首先去掉所有偶数，偶数必然不是素数，这样相当于所需要筛选的数减少了一半，存储和计算性能都得到提

高。

优化 2: 消除广播

优化 1 的代码是通过进程 0 广播下一个筛选倍数的素数。进程之间需要通过 `MPI_Bcast` 函数进行通信。通信就一定要有开销，因此我们让每个进程都各自找出它们的前 \sqrt{n} 个数中的素数，在通过这些素数筛选剩下的素数，这样一来进程之间就不需要每个循环广播素数了，性能得到提高。

优化 3: Cache 优化

在上面代码基础上，可以根据所掌握的知识，探索重构循环，提高 `cache` 命中率等方式进一步优化程序性能，并给出结果和分析。

七、实验环境：

Intel(R) Xeon(R) Gold 6342 x2

2.80GHz – 3.50GHz

24C48T x2

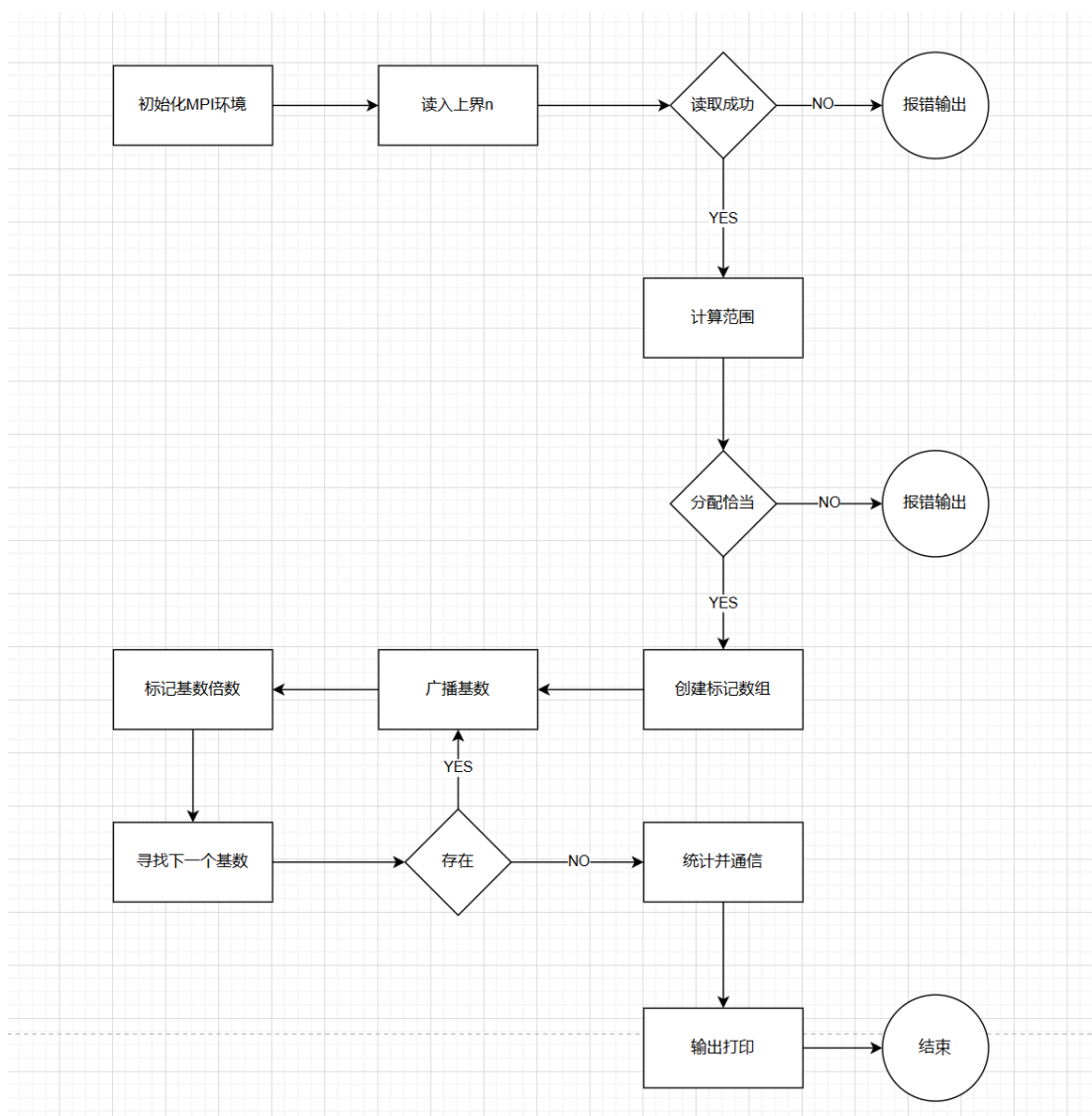
NUMA node0 CPU(s): 0-23,48-71

NUMA node1 CPU(s): 24-47,72-95

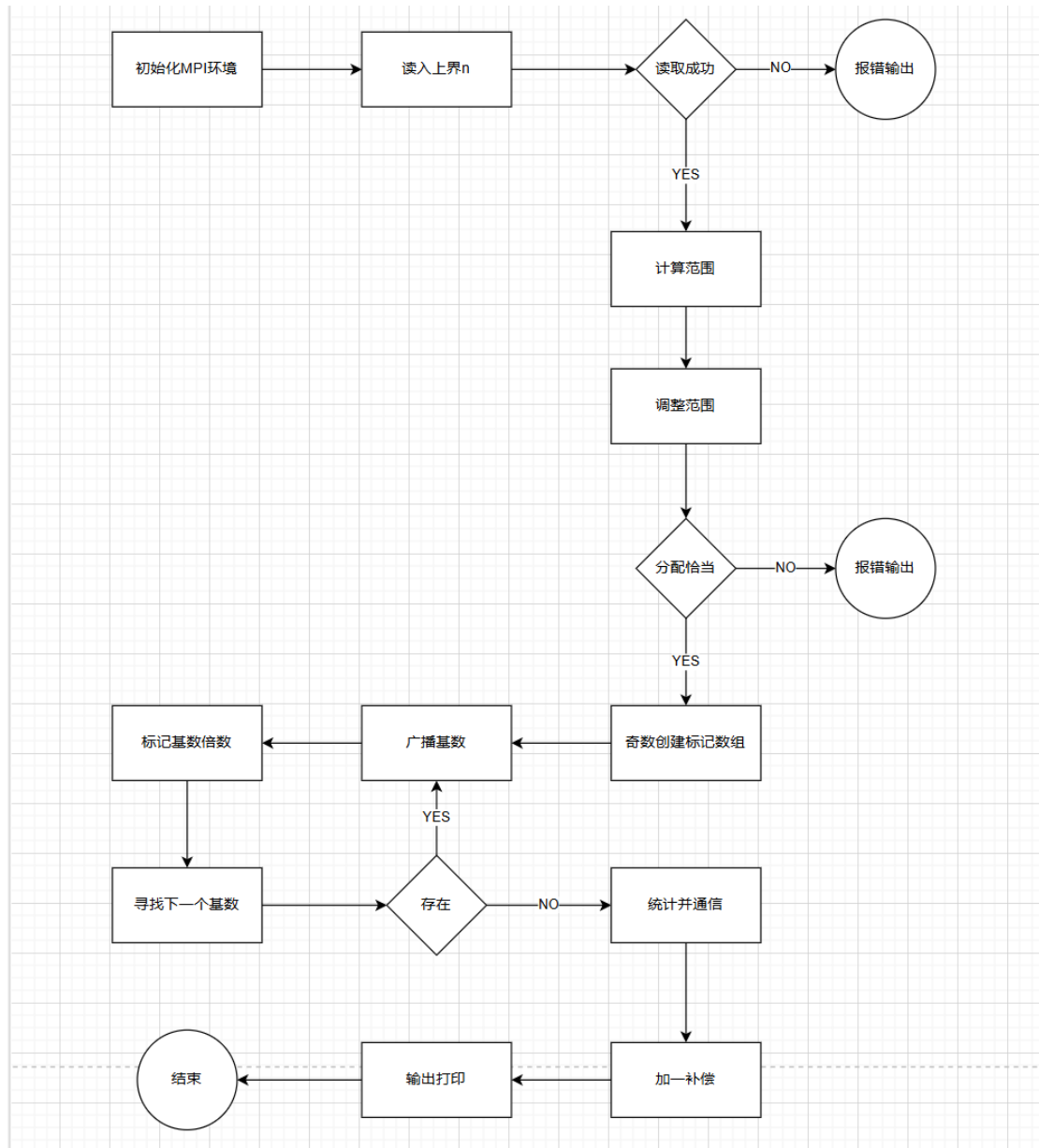
512G 内存

八、实验步骤：

1. 基准代码 base.cpp



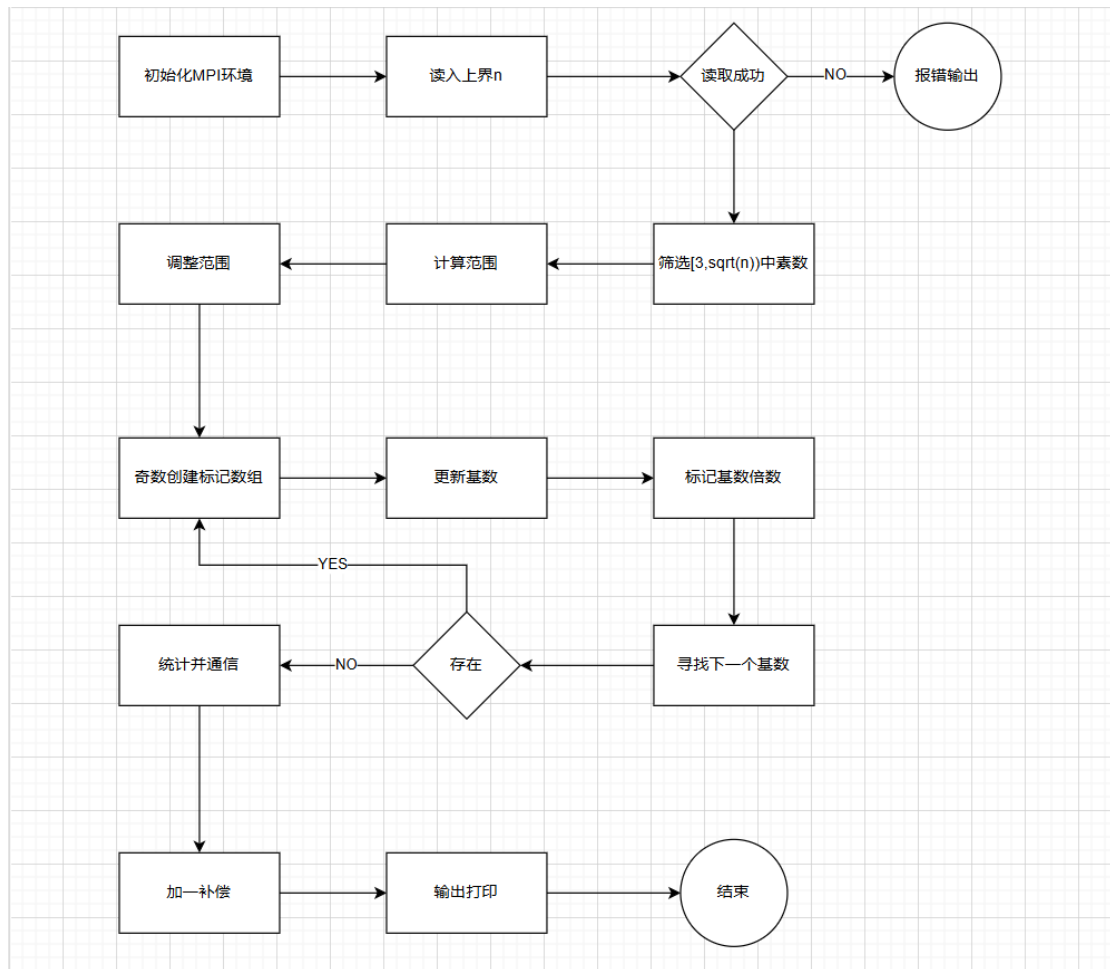
- (1) 在 cmd 中输入 `scp -P 14004 "E:\VS\MPI\MPI.cpp" a2022150501027@121.48.170.1:/home/a2022150501027` 输入密码并完成代码上传。
 - (2) 在 Xshell 中连接到服务器，输入 `mpic++ -o MPI MPI.cpp` 进行编译。
 - (3) 继续输入 `mpirun -np 1 ./MPI 1000000000` 按照规模为 1×10^9 规模并在不同进程规模 (1, 2, 4, 8, 16) 下测试三次。
2. 去除偶数 optimize1.cpp



- (1) 在 cmd 中输入 `scp -P 14004 "E:\VS\MPI\optimize1.cpp" a2022150501027@121.48.170.1:/home/a2022150501027` 输入密码并完成代码上传。
- (2) 在 Xshell 中连接到服务器，输入 `mpic++ -o optimize1 optimize1.cpp` 进行编译。
- (3) 继续输入 `mpirun -np 1 ./optimize1 1000000000` 按照规模为 1×10^9 规模并

在不同进程规模（1，2，4，8，16）下测试三次。

3. 消除广播优化 optimize2.cpp



(1) 在 cmd 中输入 `scp -P 14004`

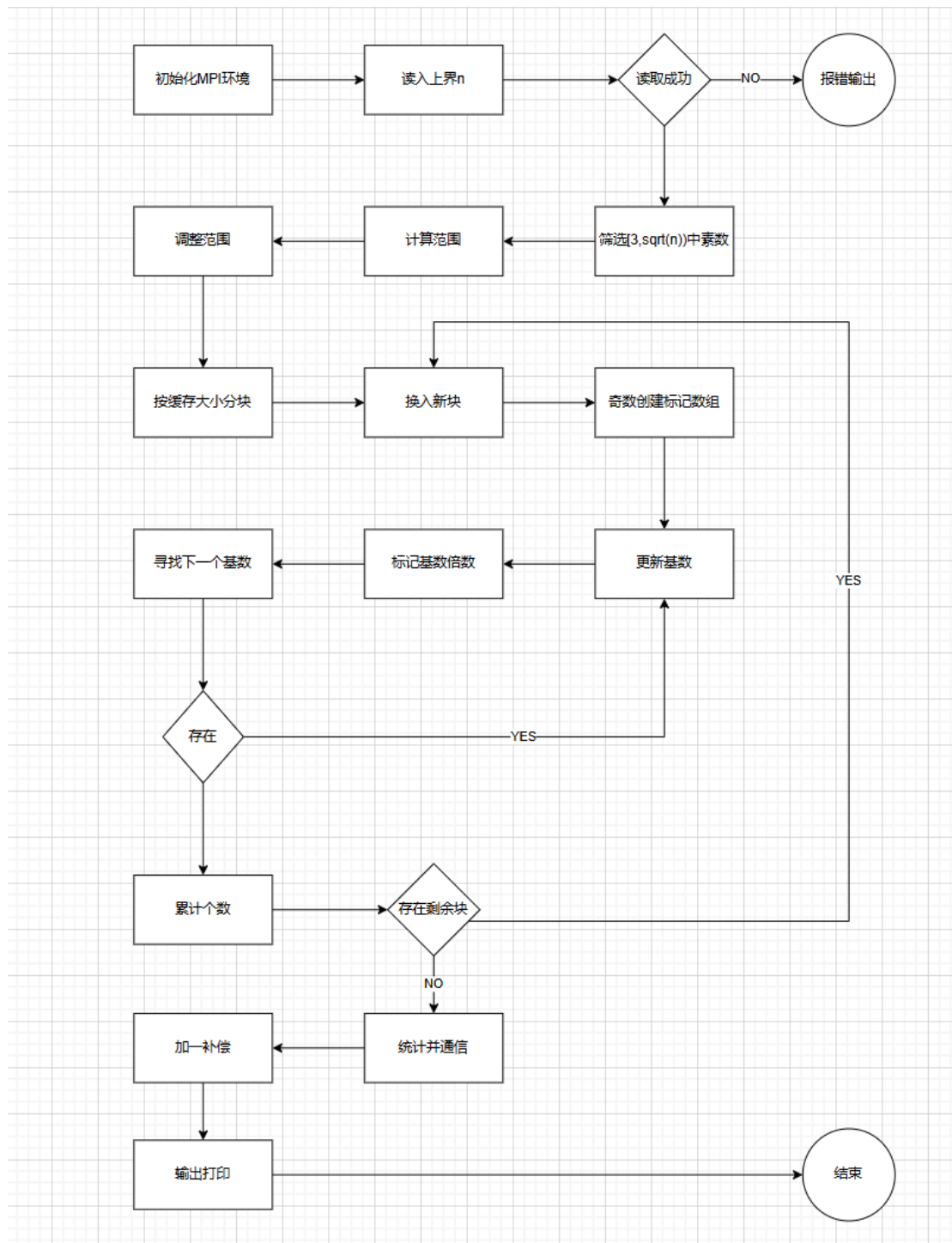
`"E:\VS\MPI\optimize2.cpp" a2022150501027@121.48.170.1:/home/a2022150501027`

输入密码并完成代码上传。

(2) 在 Xshell 中连接到服务器，输入 `mpic++ -o optimize2 optimize2.cpp` 进行编译。

(3) 继续输入 `mpirun -np 1 ./optimize2 1000000000` 按照规模为 1×10^9 规模并在不同进程规模（1，2，4，8，16）下测试三次。

4. cache 优化 optimize3.cpp



(1) 在 cmd 中输入 `scp -P 14004`

`"E:\VS\MPI\optimize3.cpp" a2022150501027@121.48.170.1:/home/a2022150501027`

输入密码并完成代码上传。

(2) 在 Xshell 中连接到服务器，输入 `mpic++ -o optimize3 optimize3.cpp` 进行编译。

(3) 继续输入 `mpirun -np 1 ./optimize3 1000000000` 按照规模为 1×10^9 规模并在不同进程规模（1，2，4，8，16）下测试三次。

4. 进一步优化 optimize4.cpp

使用 memset 替换 for 循环初始化 marked 数组

循环展开

使用 __builtin_popcountll 函数来统计

继续减少标记 marked 数组部分的分支判断

内联汇编等。

(1)在 cmd 中输入 scp -P 14004

"E:\VS\MPI\optimize4.cpp"a2022150501027@121.48.170.1:/home/a2022150501027

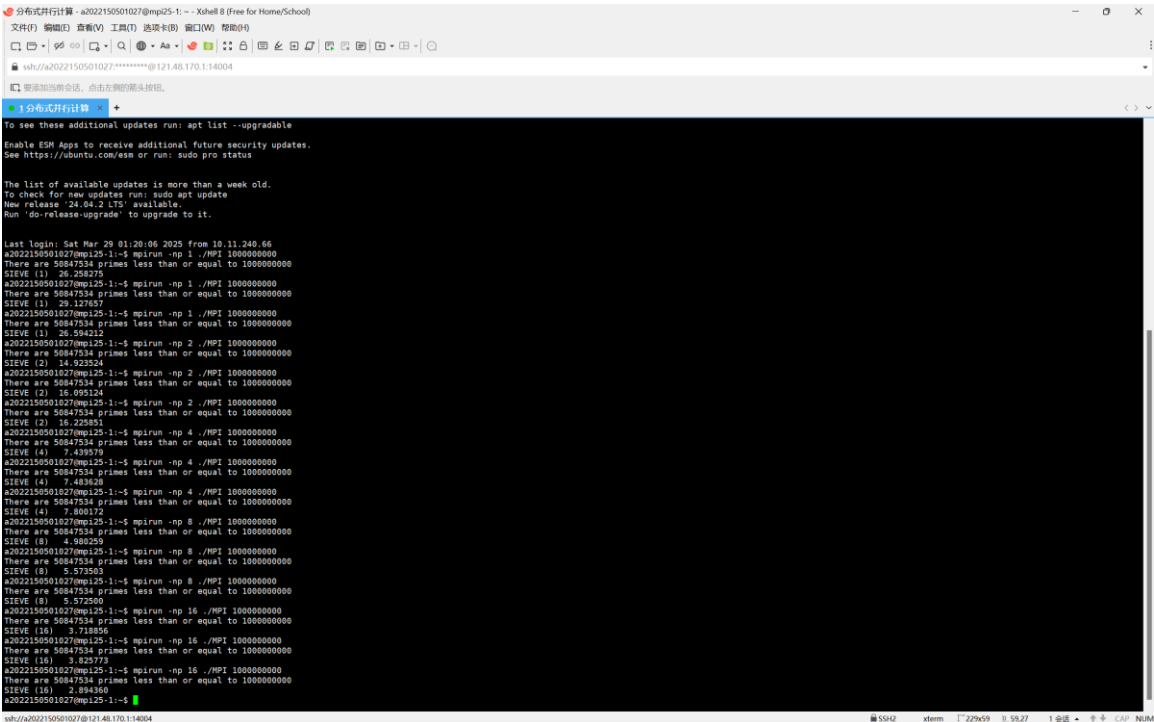
输入密码并完成代码上传。

(2)在 Xshell 中连接到服务器，输入 mpic++ -o optimize4 optimize4.cpp 进行编译。

(3)继续输入 mpirun -np 1 ./optimize4 4000000000 按照规模为 4×10^9 规模并对比其他几次优化。

九、数据分析：

1. 基准代码



次数/进程规模	1	2	4	8	16
1	26.258275	14.923524	7.439579	4.980259	3.718856
2	29.127657	16.095124	7.483628	5.573503	3.825773
3	26.594212	16.225851	7.800172	5.5725	2.89436
平均	27.3267147	15.7481663	7.57445967	5.37542067	3.479663
加速比	1	1.73523152	3.60774443	5.08364207	7.85326472

在并行程序下每个进程负责 $1/p$ 范围的筛选，理论加速比为 p 但是进程间通信以及不平衡负载降低了加速比。

2. 去除偶数

```
SIEVE (8) 73.327960
a2022150501027@mpi25-1:~$ mpirun -np 8 ./optimize1 1000000000
There are 50847534 primes less than or equal to 1000000000
SIEVE (8) 10.688815
a2022150501027@mpi25-1:~$ mpirun -np 8 ./optimize1 1000000000
There are 50847534 primes less than or equal to 1000000000
SIEVE (8) 1.664010
a2022150501027@mpi25-1:~$ mpirun -np 8 ./optimize1 1000000000
There are 50847534 primes less than or equal to 1000000000
SIEVE (8) 3.278977
a2022150501027@mpi25-1:~$ mpirun -np 8 ./optimize1 1000000000
There are 50847534 primes less than or equal to 1000000000
SIEVE (8) 2.373355
a2022150501027@mpi25-1:~$ mpirun -np 1 ./optimize1 1000000000
There are 1 primes less than or equal to 1000000000
SIEVE (1) 12.119658
a2022150501027@mpi25-1:~$ mpirun -np 1 ./optimize1 1000000000
There are 1 primes less than or equal to 1000000000
SIEVE (1) 13.795422
a2022150501027@mpi25-1:~$ mpirun -np 1 ./optimize1 1000000000
There are 1 primes less than or equal to 1000000000
SIEVE (1) 12.120343
a2022150501027@mpi25-1:~$ mpirun -np 2 ./optimize1 1000000000
There are 50847534 primes less than or equal to 1000000000
SIEVE (2) 6.954315
a2022150501027@mpi25-1:~$ mpirun -np 2 ./optimize1 1000000000
There are 50847534 primes less than or equal to 1000000000
SIEVE (2) 7.048599
a2022150501027@mpi25-1:~$ mpirun -np 2 ./optimize1 1000000000
There are 50847534 primes less than or equal to 1000000000
SIEVE (2) 6.162272
a2022150501027@mpi25-1:~$ mpirun -np 4 ./optimize1 1000000000
There are 50847534 primes less than or equal to 1000000000
SIEVE (4) 3.978210
a2022150501027@mpi25-1:~$ mpirun -np 4 ./optimize1 1000000000
There are 50847534 primes less than or equal to 1000000000
SIEVE (4) 4.544250
a2022150501027@mpi25-1:~$ mpirun -np 4 ./optimize1 1000000000
There are 50847534 primes less than or equal to 1000000000
SIEVE (4) 3.426197
a2022150501027@mpi25-1:~$ mpirun -np 8 ./optimize1 1000000000
There are 50847534 primes less than or equal to 1000000000
SIEVE (8) 1.988108
a2022150501027@mpi25-1:~$ mpirun -np 8 ./optimize1 1000000000
There are 50847534 primes less than or equal to 1000000000
SIEVE (8) 2.475250
a2022150501027@mpi25-1:~$ mpirun -np 8 ./optimize1 1000000000
There are 50847534 primes less than or equal to 1000000000
SIEVE (8) 1.939329
a2022150501027@mpi25-1:~$ mpirun -np 16 ./optimize1 1000000000
There are 50847534 primes less than or equal to 1000000000
SIEVE (16) 1.260829
a2022150501027@mpi25-1:~$ mpirun -np 16 ./optimize1 1000000000
There are 50847534 primes less than or equal to 1000000000
SIEVE (16) 1.386308
a2022150501027@mpi25-1:~$ mpirun -np 16 ./optimize1 1000000000
There are 50847534 primes less than or equal to 1000000000
SIEVE (16) 1.470149
a2022150501027@mpi25-1:~$
```

次数/进程规模	1	2	4	8	16
1	12.119658	6.954315	3.97821	1.988108	1.260829
2	13.795422	7.048599	4.54425	2.47525	1.386308
3	12.120343	6.162272	3.426197	1.939329	1.470149
平均	12.6784743	6.72172867	3.98288567	2.134229	1.37242867
加速比	1	1.88619252	3.18323833	5.94054074	9.2379842

每个进程对自己的上下界进行优化并且减少了进程 0 的工作负载。提高了加速比的理论值。

3. 消除广播优化

```
There are 50847534 primes less than or equal to 1000000000
SIEVE (16) 2.894360
a2022150501027@mpi25-1:~$ mpirun -np 8 ./optimize2 1000000000
a2022150501027@mpi25-1:~$ mpirun -np 8 ./optimize2 1000000000
There are 1 primes less than or equal to 1000000000
SIEVE (1) 14.687564
a2022150501027@mpi25-1:~$ mpirun -np 1 ./optimize2 1000000000
There are 1 primes less than or equal to 1000000000
SIEVE (1) 12.939469
a2022150501027@mpi25-1:~$ mpirun -np 1 ./optimize2 1000000000
There are 1 primes less than or equal to 1000000000
SIEVE (1) 12.945740
a2022150501027@mpi25-1:~$ mpirun -np 2 ./optimize2 1000000000
There are 50847534 primes less than or equal to 1000000000
SIEVE (2) 7.371873
a2022150501027@mpi25-1:~$ mpirun -np 2 ./optimize2 1000000000
There are 50847534 primes less than or equal to 1000000000
SIEVE (2) 7.966230
a2022150501027@mpi25-1:~$ mpirun -np 2 ./optimize2 1000000000
There are 50847534 primes less than or equal to 1000000000
SIEVE (2) 6.618769
a2022150501027@mpi25-1:~$ mpirun -np 4 ./optimize2 1000000000
There are 50847534 primes less than or equal to 1000000000
SIEVE (4) 3.628448
a2022150501027@mpi25-1:~$ mpirun -np 4 ./optimize2 1000000000
There are 50847534 primes less than or equal to 1000000000
SIEVE (4) 3.772963
a2022150501027@mpi25-1:~$ mpirun -np 4 ./optimize2 1000000000
There are 50847534 primes less than or equal to 1000000000
SIEVE (4) 4.436891
a2022150501027@mpi25-1:~$ mpirun -np 8 ./optimize2 1000000000
There are 50847534 primes less than or equal to 1000000000
SIEVE (8) 2.329726
a2022150501027@mpi25-1:~$ mpirun -np 8 ./optimize2 1000000000
There are 50847534 primes less than or equal to 1000000000
SIEVE (8) 2.375938
a2022150501027@mpi25-1:~$ mpirun -np 8 ./optimize2 1000000000
There are 50847534 primes less than or equal to 1000000000
SIEVE (8) 2.220610
a2022150501027@mpi25-1:~$ mpirun -np 16 ./optimize2 1000000000
There are 50847534 primes less than or equal to 1000000000
SIEVE (16) 1.654669
a2022150501027@mpi25-1:~$ mpirun -np 16 ./optimize2 1000000000
There are 50847534 primes less than or equal to 1000000000
SIEVE (16) 3.033654
a2022150501027@mpi25-1:~$ mpirun -np 16 ./optimize2 1000000000
There are 50847534 primes less than or equal to 1000000000
SIEVE (16) 3.089312
a2022150501027@mpi25-1:~$ mpirun -np 16 ./optimize2 4000000000
There are 189961612 primes less than or equal to 4000000000
SIEVE (16) 5.014189
a2022150501027@mpi25-1:~$ mpirun -np 16 ./optimize2 4000000000
There are 1 primes less than or equal to 4000000000
SIEVE (1) 51.822793
a2022150501027@mpi25-1:~$ mpirun -np 4 ./optimize2 4000000000
There are 189961612 primes less than or equal to 4000000000
SIEVE (4) 13.858738
a2022150501027@mpi25-1:~$
```


次数/进程规模	1	2	4	8	16
1	14.667564	7.371873	3.629448	2.329728	1.054603
2	12.939469	7.966023	3.772963	2.375918	1.013994
3	12.94574	6.618789	4.436691	2.22061	1.089512
平均	13.517591	7.318895	3.94636733	2.308752	1.052703
加速比	1	1.84694424	3.42532508	5.85493418	12.8408402

广播消除加速了进程本身也让进程负载更加均衡。加速比提升更加显著

4. cache 优化

次数/进程规模	1	2	4	8	16
1	3.774396	2.156237	1.04925	0.604573	0.449574
2	3.744088	1.975542	1.058412	0.672635	0.452966
3	3.729113	1.967953	1.076838	0.601009	0.444838
平均	3.749199	2.033244	1.0615	0.62607233	0.449126
加速比	1	1.84394937	3.5319821	5.98844383	8.34776655

次数/进程规模	1	2	4	8	16
1	3.774396	2.156237	1.04925	0.604573	0.449574
2	3.744088	1.975542	1.058412	0.672635	0.452966
3	3.729113	1.967953	1.076838	0.601009	0.444838
平均	3.749199	2.033244	1.0615	0.62607233	0.449126
加速比	1	1.84394937	3.5319821	5.98844383	8.34776655

显著提升了程序性能

5. 进一步优化

次数/进程规模	1	2	4	8	16
1	3.774396	2.156237	1.04925	0.604573	0.449574
2	3.744088	1.975542	1.058412	0.672635	0.452966
3	3.729113	1.967953	1.076838	0.601009	0.444838
平均	3.749199	2.033244	1.0615	0.62607233	0.449126
加速比	1	1.84394937	3.5319821	5.98844383	8.34776655

相比于 optimize2 和 optimize3 来说，在进程规模为 8，问题规模为 4×10^9 下进

一步提高了性能。

十、实验结论：

三次优化后均能正确计算出素数个数，并且每一次优化都能极大加快进程的性能。整体运行时间大幅度缩短，加速比更加贴近理论值。通过 MPI 并行编程大大提高了算法的效率。

十一、总结及心得体会：

MPI 并行编程能够大大提高特定问题范围内的求解性能，并且随着进程规模提高，加速比大致呈线性增长。此次实验使我对 MPI 的应用有了更加深刻的理解，了解到了埃拉托斯特尼筛选法的思想。通过三次优化和调试，理解了并行程序的问题和特点。

朱若久