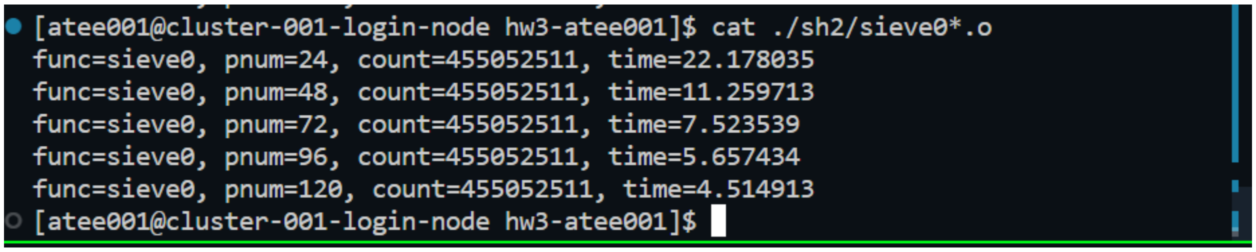
Name: Andrew Tee
Github Account Name: atee001
Github Repo for HW3: https://github.com/CS211-Fall2023/hw3-atee001

HW# 3

## PROBLEM 1

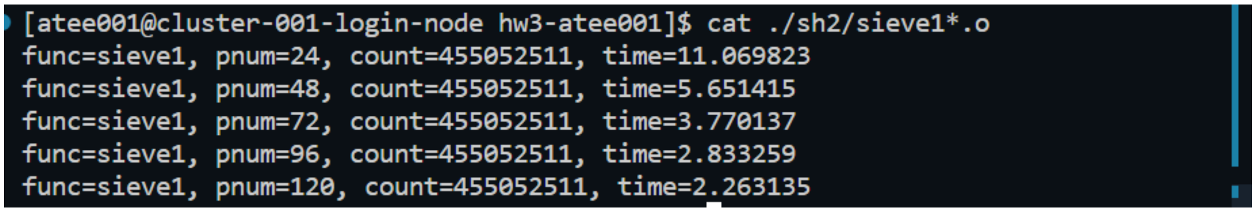
Image shows sieve0 runtime and total prime number count for pnum = 24, 48, 72, 96, 120



```
[atee001@cluster-001-login-node hw3-atee001]$ cat ./sh2/sieve0*.o
func=sieve0, pnum=24, count=455052511, time=22.178035
func=sieve0, pnum=48, count=455052511, time=11.259713
func=sieve0, pnum=72, count=455052511, time=7.523539
func=sieve0, pnum=96, count=455052511, time=5.657434
func=sieve0, pnum=120, count=455052511, time=4.514913
[atee001@cluster-001-login-node hw3-atee001]$
```

## PROBLEM 2

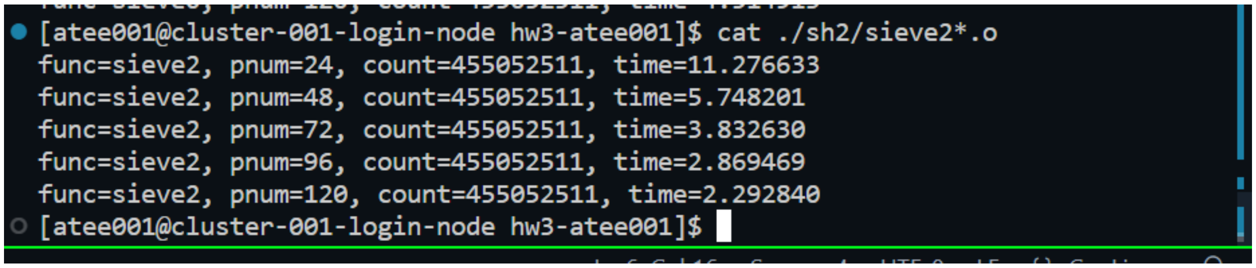
Output for pnum = 24, 48, 72, 96 120



```
[atee001@cluster-001-login-node hw3-atee001]$ cat ./sh2/sieve1*.o
func=sieve1, pnum=24, count=455052511, time=11.069823
func=sieve1, pnum=48, count=455052511, time=5.651415
func=sieve1, pnum=72, count=455052511, time=3.770137
func=sieve1, pnum=96, count=455052511, time=2.833259
func=sieve1, pnum=120, count=455052511, time=2.263135
```

## PROBLEM 3

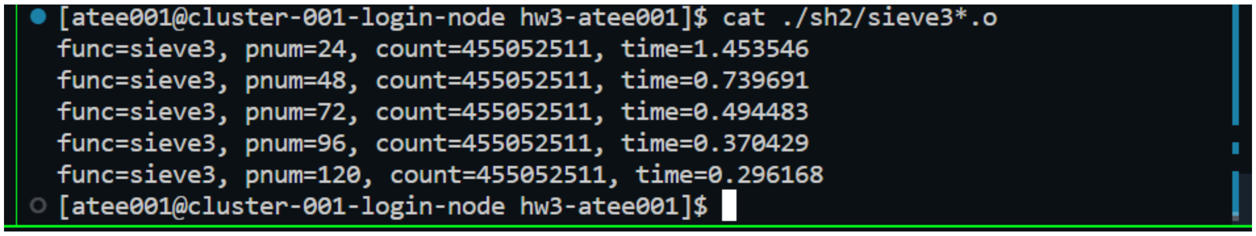
Output for pnum = 24, 48, 72, 96 120



```
[atee001@cluster-001-login-node hw3-atee001]$ cat ./sh2/sieve2*.o
func=sieve2, pnum=24, count=455052511, time=11.276633
func=sieve2, pnum=48, count=455052511, time=5.748201
func=sieve2, pnum=72, count=455052511, time=3.832630
func=sieve2, pnum=96, count=455052511, time=2.869469
func=sieve2, pnum=120, count=455052511, time=2.292840
[atee001@cluster-001-login-node hw3-atee001]$
```

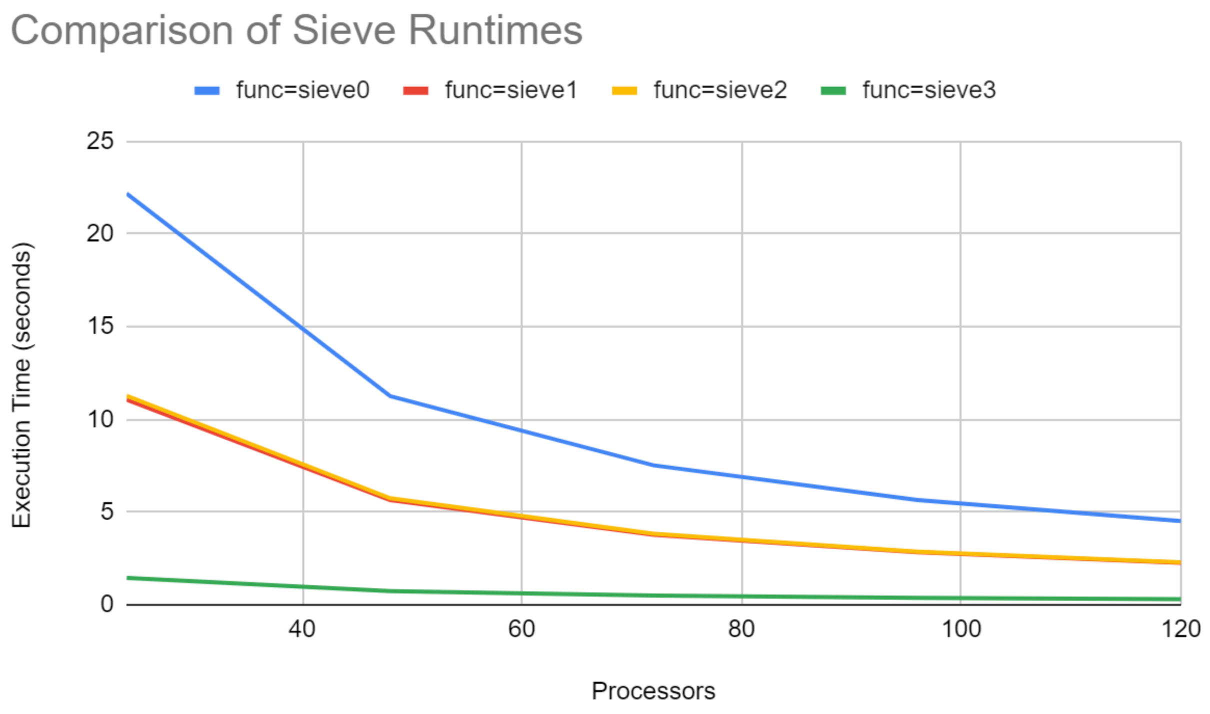
## PROBLEM 4

Output for pnum = 24, 48, 72, 96 120



```
[atee001@cluster-001-login-node hw3-atee001]$ cat ./sh2/sieve3*.o
func=sieve3, pnum=24, count=455052511, time=1.453546
func=sieve3, pnum=48, count=455052511, time=0.739691
func=sieve3, pnum=72, count=455052511, time=0.494483
func=sieve3, pnum=96, count=455052511, time=0.370429
func=sieve3, pnum=120, count=455052511, time=0.296168
[atee001@cluster-001-login-node hw3-atee001]$
```

Execution time for all sieves. Figure demonstrates an inversely proportional relationship with execution time and processors.



## Analysis

| Processes | func=sieve0 | func=sieve1 | func=sieve2 | func=sieve3 |
|---|---|---|---|---|
| 24 | 22.178035 | 11.069823 | 11.276633 | 1.453546 |
| 48 | 11.259713 | 5.651415 | 5.748201 | 0.739691 |
| 72 | 7.523539 | 3.770137 | 3.83263 | 0.494483 |
| 96 | 5.657434 | 2.833259 | 2.869469 | 0.370429 |
| 120 | 4.514913 | 2.263135 | 2.29284 | 0.296168 |

Q. Check whether your execution time is inversely proportional to the number of cores.

The relationship between execution time and the number of processors appears to be inversely proportional. This trend is particularly noticeable in the case of sieve1 and sieve2, where the graph exhibits a clear inverse correlation.

1. Sieve 1 vs Sieve 0

The execution time of sieve1 is half that of sieve0. This aligns with theoretical expectations, as eliminating all even numbers reduces the size of the list by half, resulting in only half the number of numbers to process.

2. Sieve 2 vs Sieve 1

The execution time of sieve1 and sieve2 shows no significant differences. The theoretical expectation was that removing the broadcast between different processors should accelerate the execution time. However, without the broadcast, each processor must independently compute its own sieving primes, leading to an increase in the total number of computations per processor (except for processor 0). The overhead of calculating sieving primes does not seem to provide a significant speedup compared to the broadcast time. It's worth noting that the total number of processors is 120, and as the number of processors increases, the overall communication overhead may surpass the potential speedup gained from having each processor find its sieving primes, particularly if pnum continues to increase. As the number of processors (pnum) increases, the communication overhead also rises. However, if the value of n (10^10) remains constant, the time spent on sieving prime computation for each processor remains consistent. Therefore as pnum increases past 120, sieve2 should become faster than sieve1.

3. Sieve 3 vs Sieve 2

Optimizing cache usage, specifically minimizing cache misses during array marking, leads to a significant improvement in the execution time of sieve2. The speedup achieved is over sevenfold when comparing sieve3 to sieve2, where speedup is defined as the ratio of the execution time of sieve2 to the execution time of sieve3.

| **func=sieve2** | **func=sieve3** | **Speed up of Sieve3 vs Sieve2** |
|---:|---:|:---|
| 11.276633 | 1.453546 | 7.758015914 |
| 5.748201 | 0.739691 | 7.771084142 |
| 3.83263 | 0.494483 | 7.75078213 |
| 2.869469 | 0.370429 | 7.746340054 |
| 2.29284 | 0.296168 | 7.741687151 |