PRIME FACTORIZATION

Shubham Ambavale

sambaval@buffalo.edu

GUIDED BY : Dr. Russ Miller

University at Buffalo The State University of New York

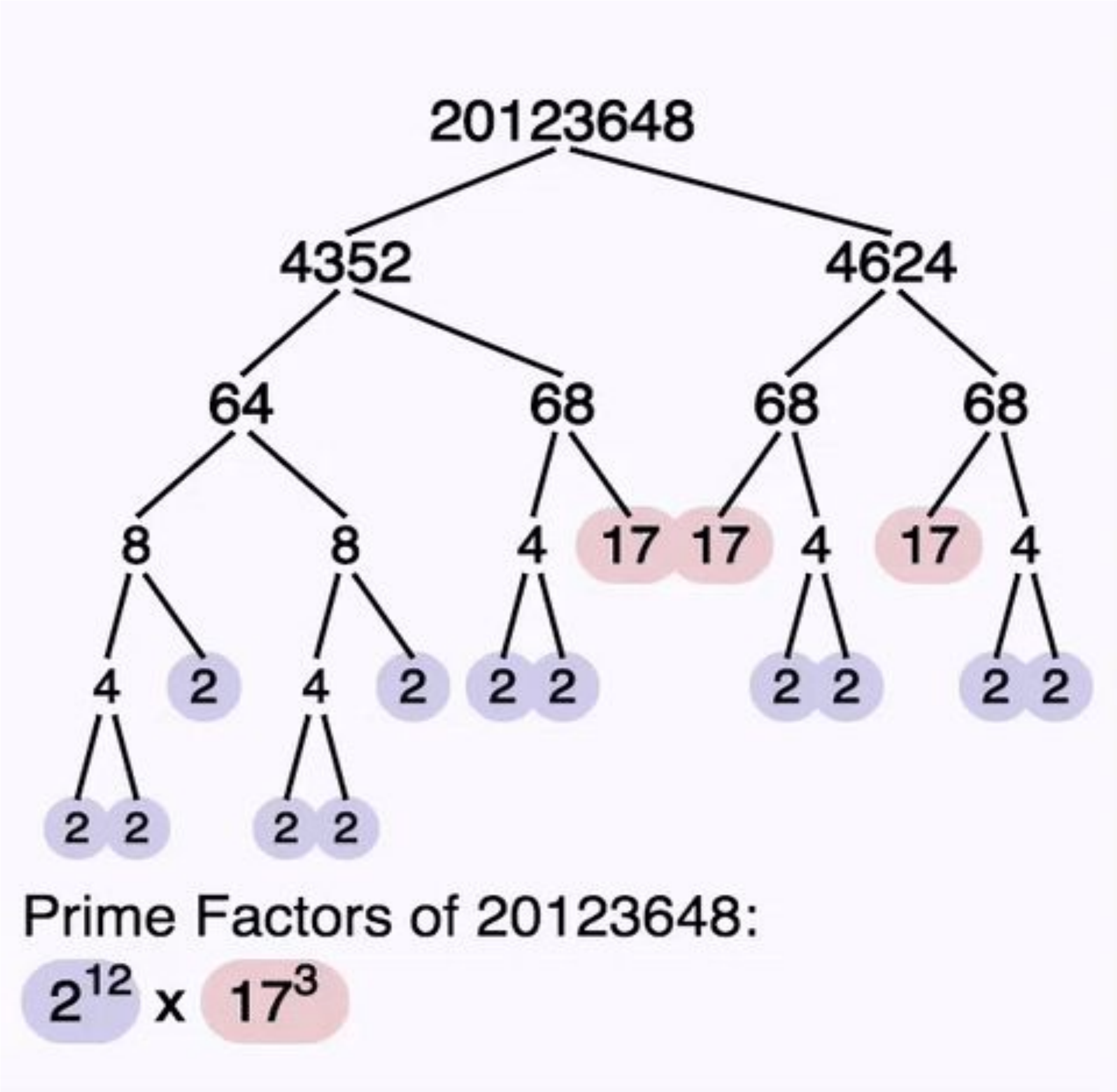
Background

- Prime factorization is the process of breaking down a composite number into its prime factors, which are the prime numbers that multiply together to equal the original number.
- Some of the applications are :

  - Cryptography(*RSA, Computer Security*)
  - Physics(Study properties of materials & their electronic structures)
  - Database Design(Unique Identifiers)
  - Optimization Problems(Computationally Intensive)
  - Cryptocurrency(Proof-Of-Work System)

Naive Sequential Algorithm

- Iterate through 2 to n
- Divide the number n until its evenly divisible.
- Evenly divisible number is one of the factor of the number n.



WORKING -

Sieve(n):

```
Input: an integer n > 1.

Let A be an array of Boolean values, indexed by integers 2 to n,
initially all set to true.

for i = 2, 3, 4, ..., not exceeding √n:
  if A[i] is true:
    for j = i², i²+i, i²+2i, i²+3i, ..., not exceeding n:
      A[j] := false.

Output: all i such that A[i] is true.
```

```
primes = Sieve(n)
for primeNumber in primeNumbers:
    while n is divisible by primeNumber:
        add primeNumber to the factor list
        divide n by primeNumber and update
```

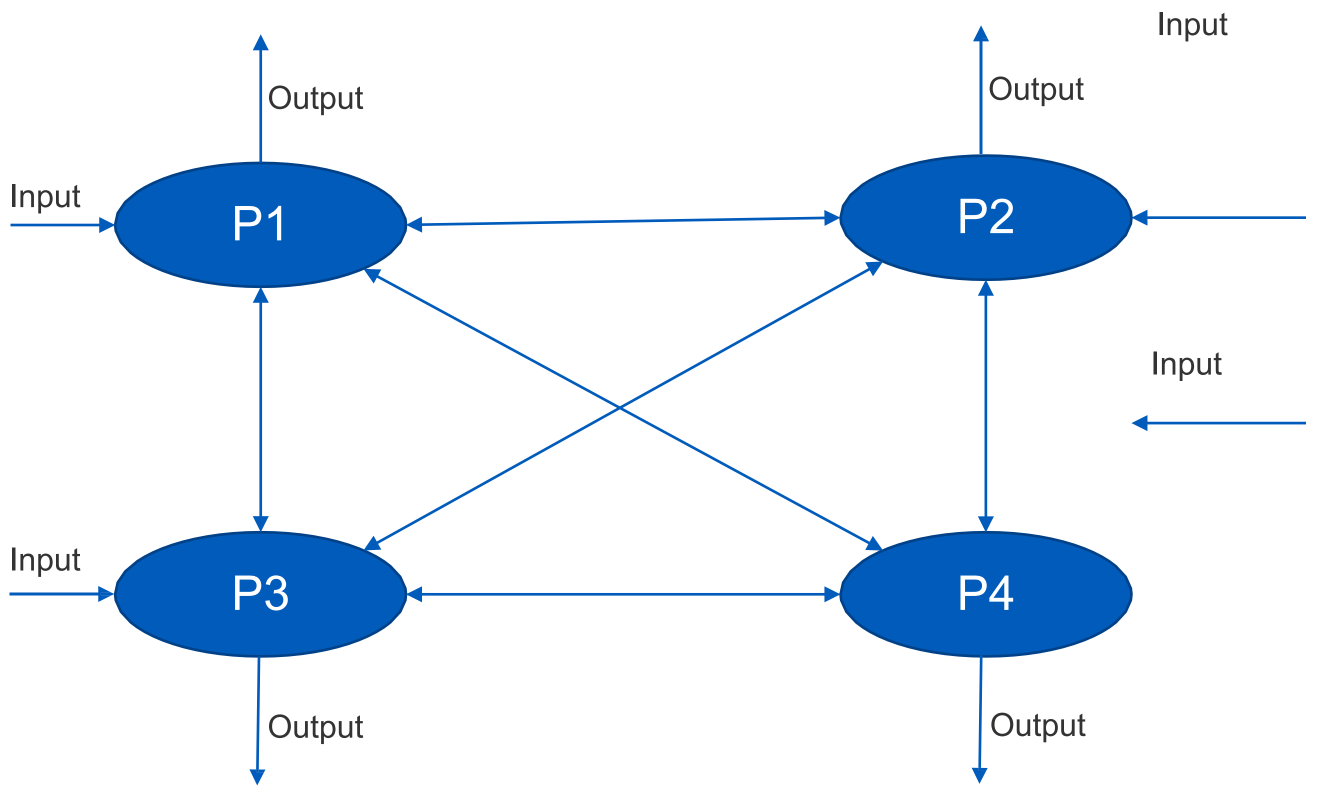

4

NEED FOR PARALLELIZATION

- Improve Factorization Performance
- Faster Execution time
- Scaling for larger inputs

5

PARALLELIZATION STRATEGY



6

PARALLELIZATION STRATEGY

- Parallelize sieve to find prime factors
- Then, distribute the primes equally into number of processors except the last processor
- Machine processes only the subset of the primes and find factors from them

7

PARALLEL ALGORITHM

- Divide input range across all the processors
- Apply sieve by marking multiples of prime in the range
- On completion, broadcast prime number to other processors to eliminate non-primes
- Consolidate(MPI_Reduce)
- Distribute the primes equally across the processors
  - n^{th} rank * chunksize <= nth processor < n^{th} rank * chunksize + chunksize
- Find prime factors which evenly divide **n** in the range
- Terminate

Improvements/Observations

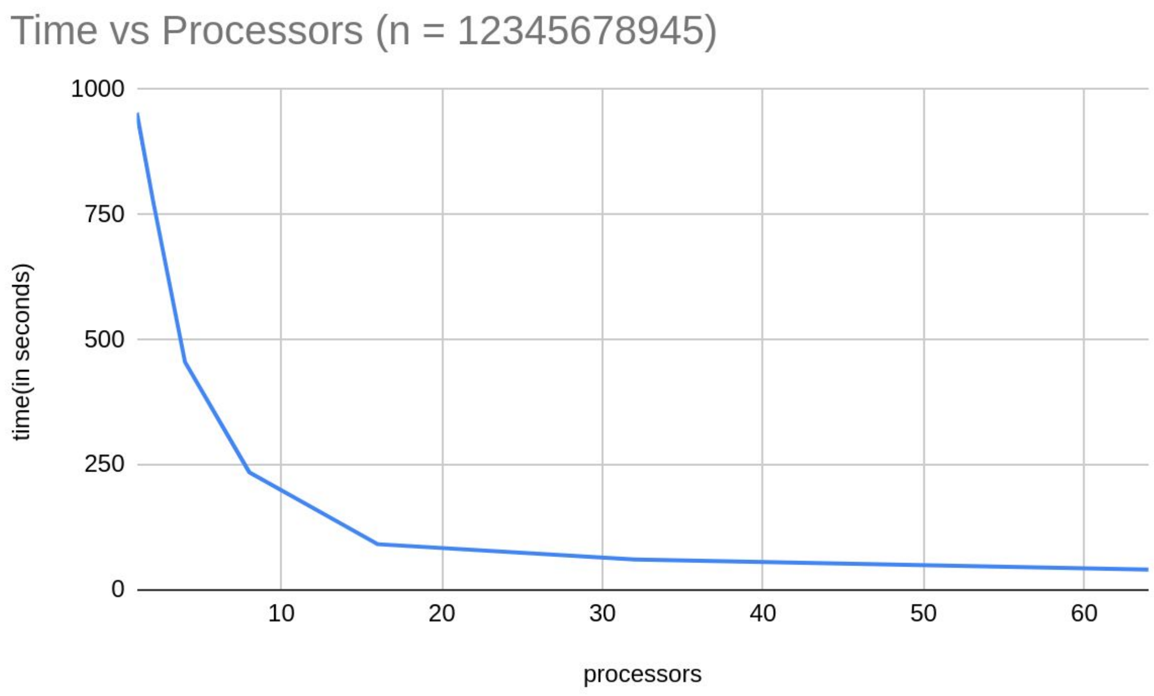
- Parallelized Sieve of Eratosthenes(Previous Bottleneck) ✅
- Scaled up the number of nodes✅
  - Increase in input size and increase in number of nodes/processors is directly proportional w.r.t performance
  - Decrease in input size and increase in number of nodes/processors is inversely proportional w.r.t performance
- Implement Lowest Prime Factor for per processor optimization (LPF) ❌
  - LPF[18] = 2 => 18/2 = 9;
  - LPF[9] = 3 => 9/3 = 3;
  - LPF[3] = 3 => 3/3 = 1;

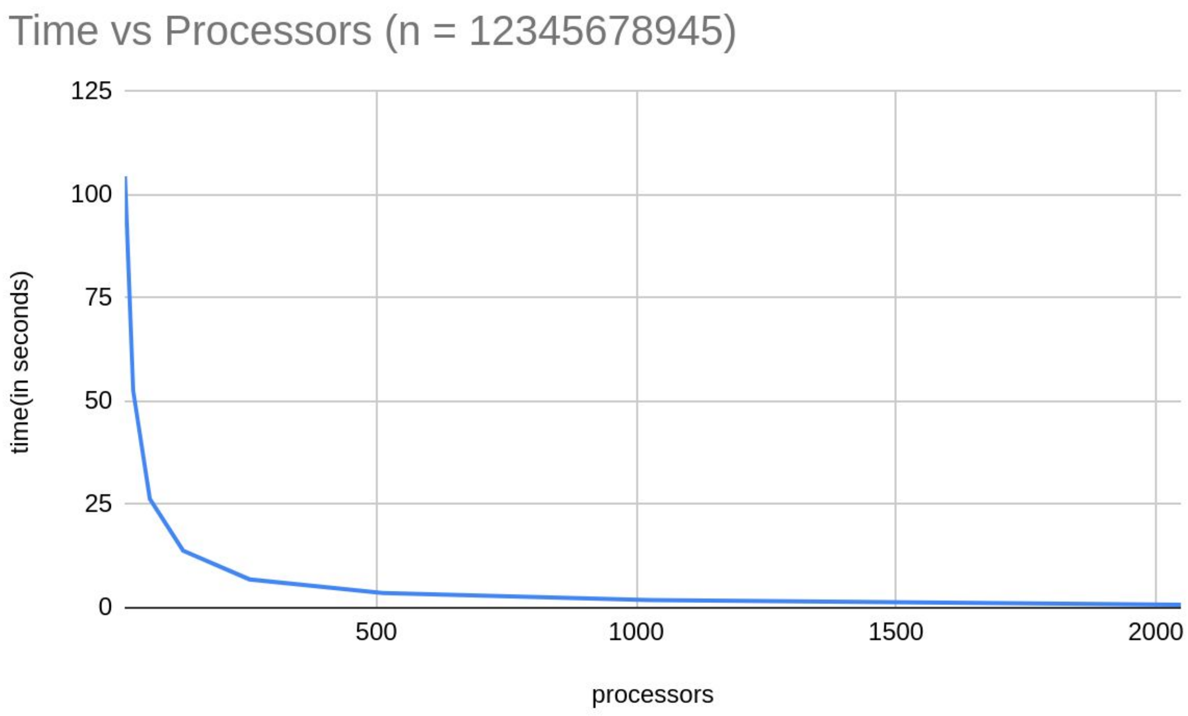
Results with equal distribution of prime factors

Processors	Time(in seconds)
1	953.654
2	779.814
4	455.916
8	234.902
16	91.448
32	60.899
64	40.548

Processors	Time(in seconds)
16	104.503741
32	52.600724
64	26.33962
128	13.821954
256	6.867855
512	3.586585
1024	1.898545
2048	0.756428

Results with equal distribution of prime factors



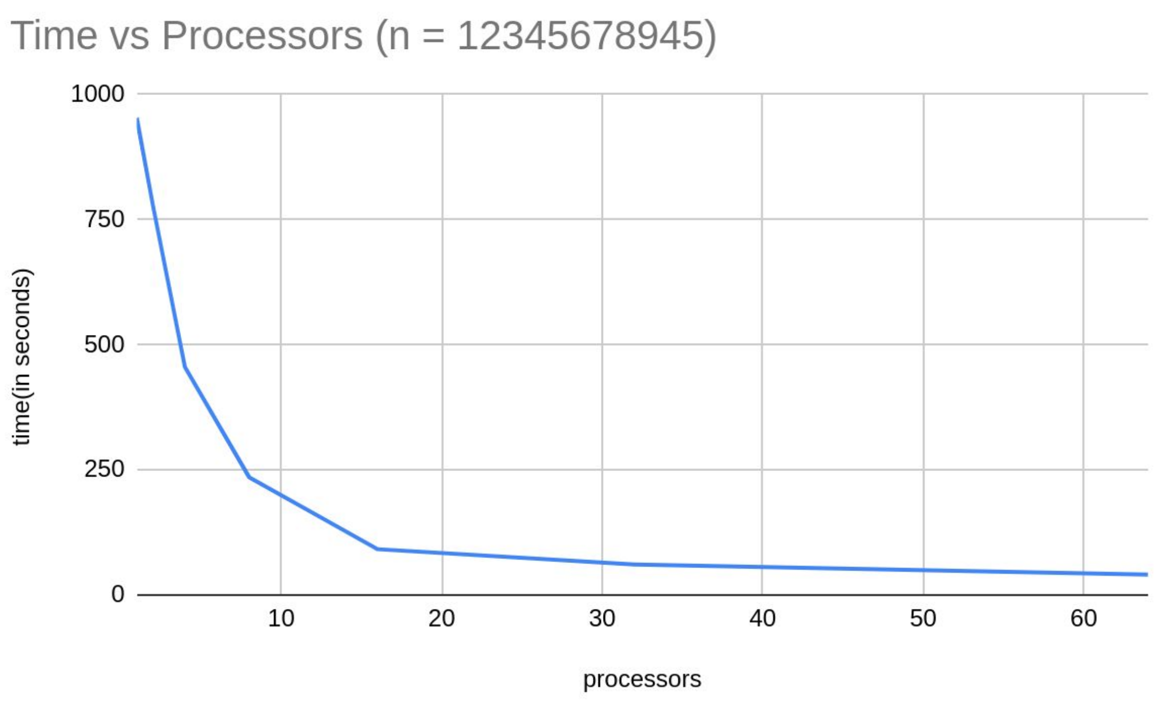


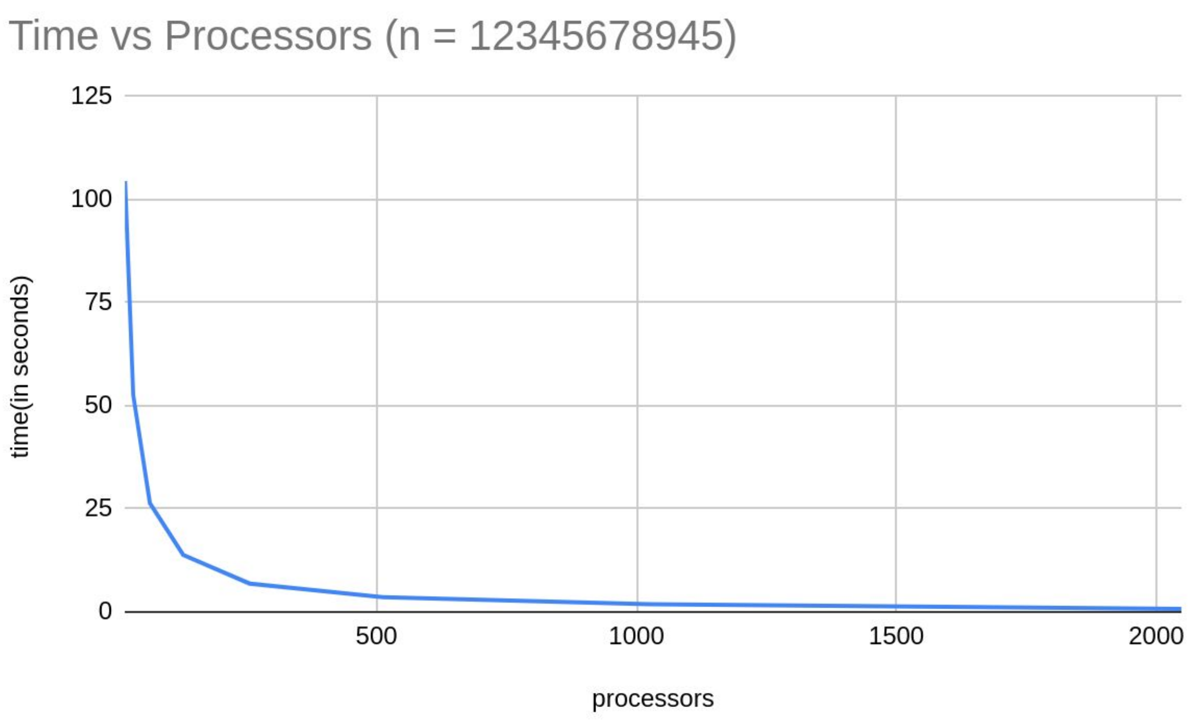
Results with exponential distribution of prime factors

Processors	Time(in seconds)
1	1023.654
2	800.814
4	480.655
8	290.659
16	150.598
32	100.665
64	80.655

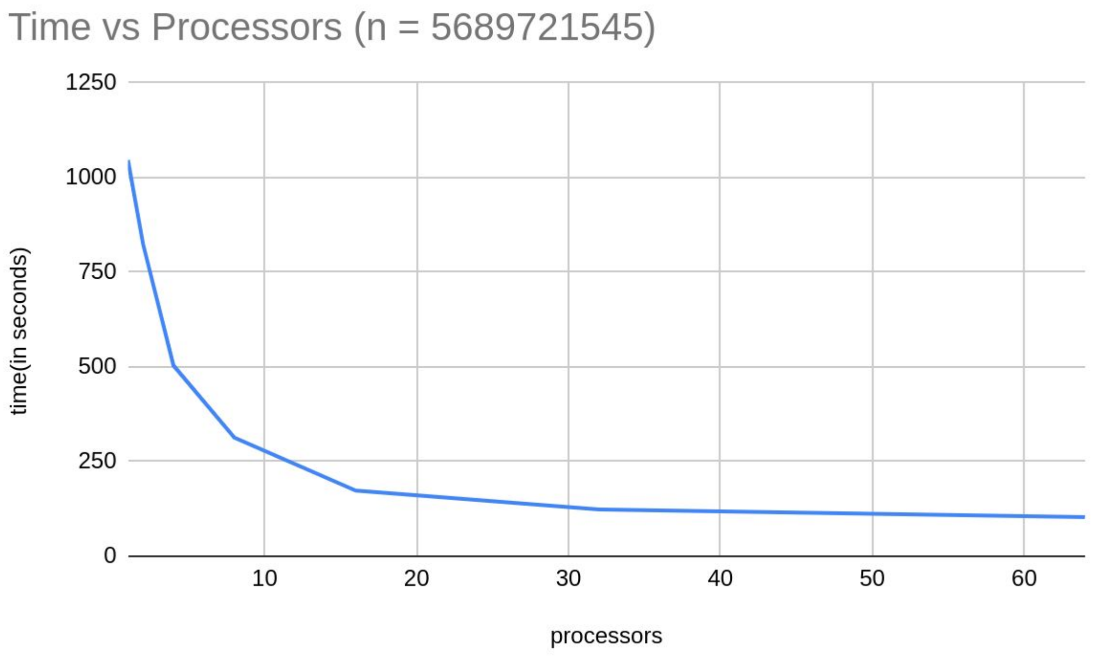
Processors	Time(in seconds)
16	114.598
32	60.598
64	40.566
128	25.526
256	15.565
512	10.889
1024	5.233
2048	2.265

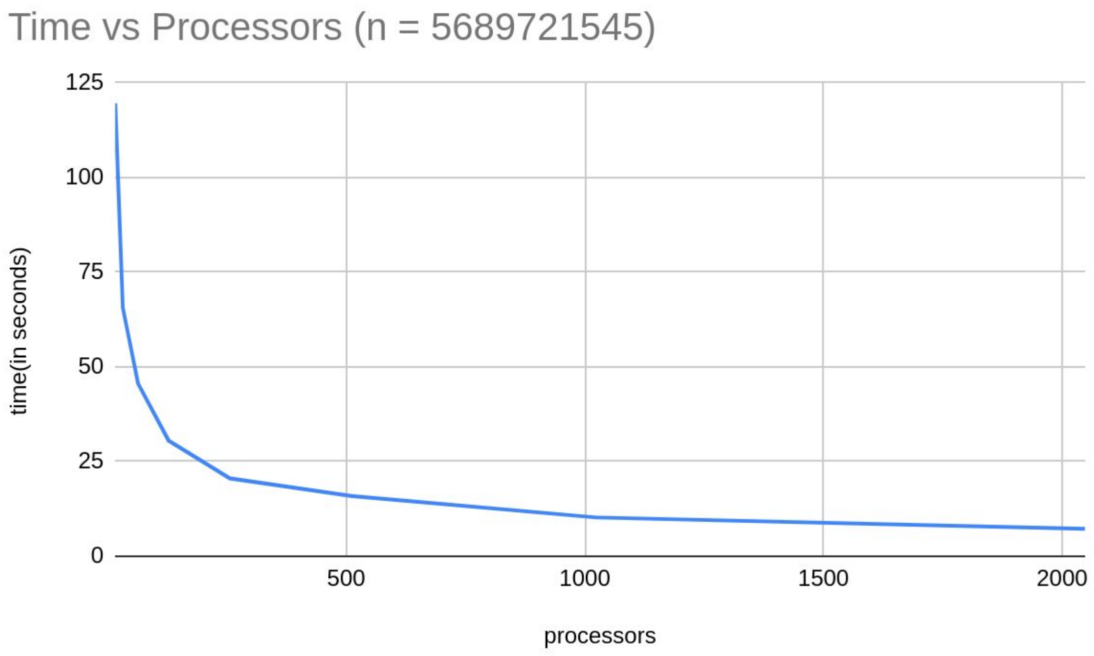
Results with exponential distribution of prime factors



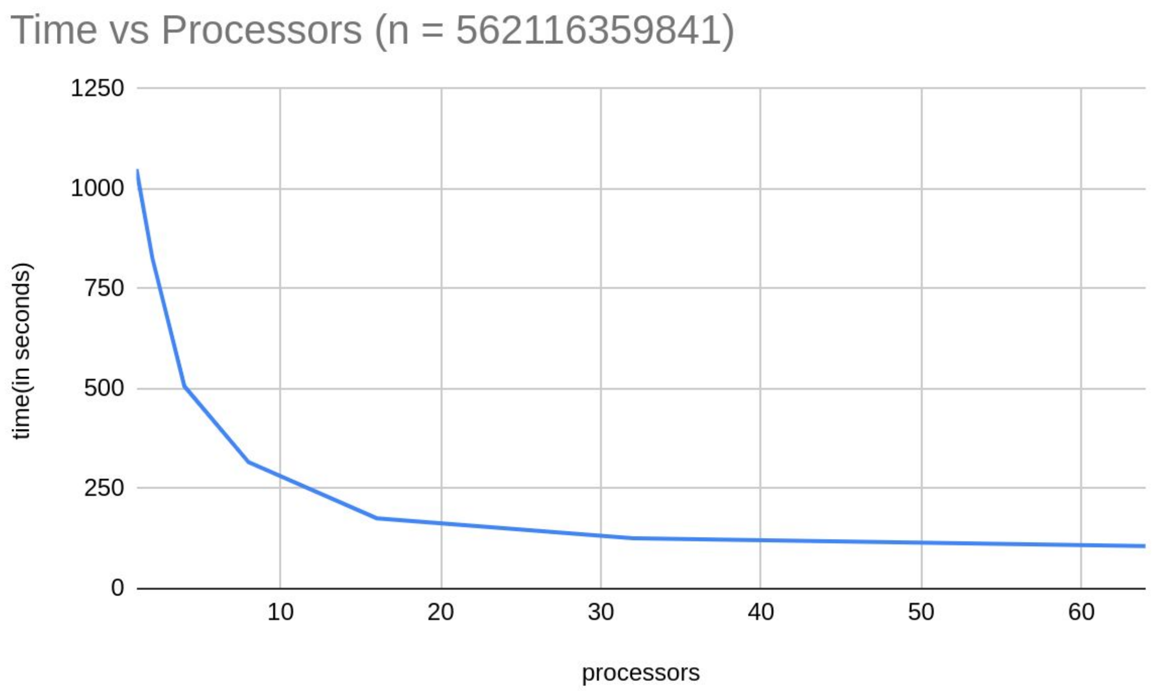


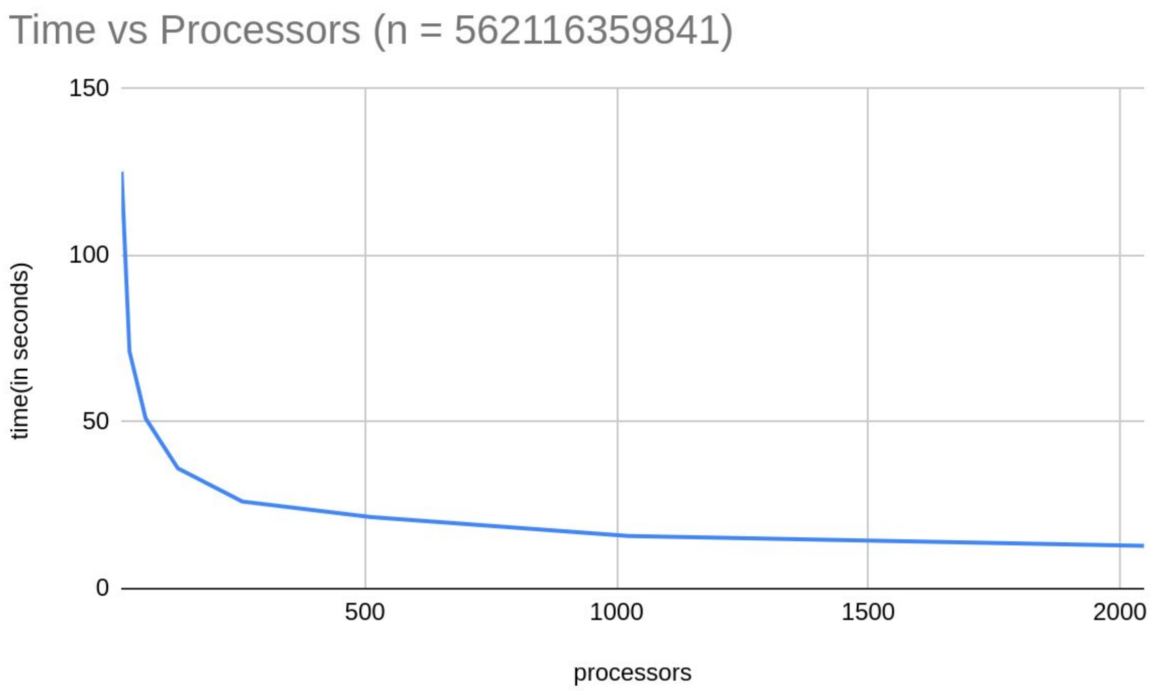
Variation for n



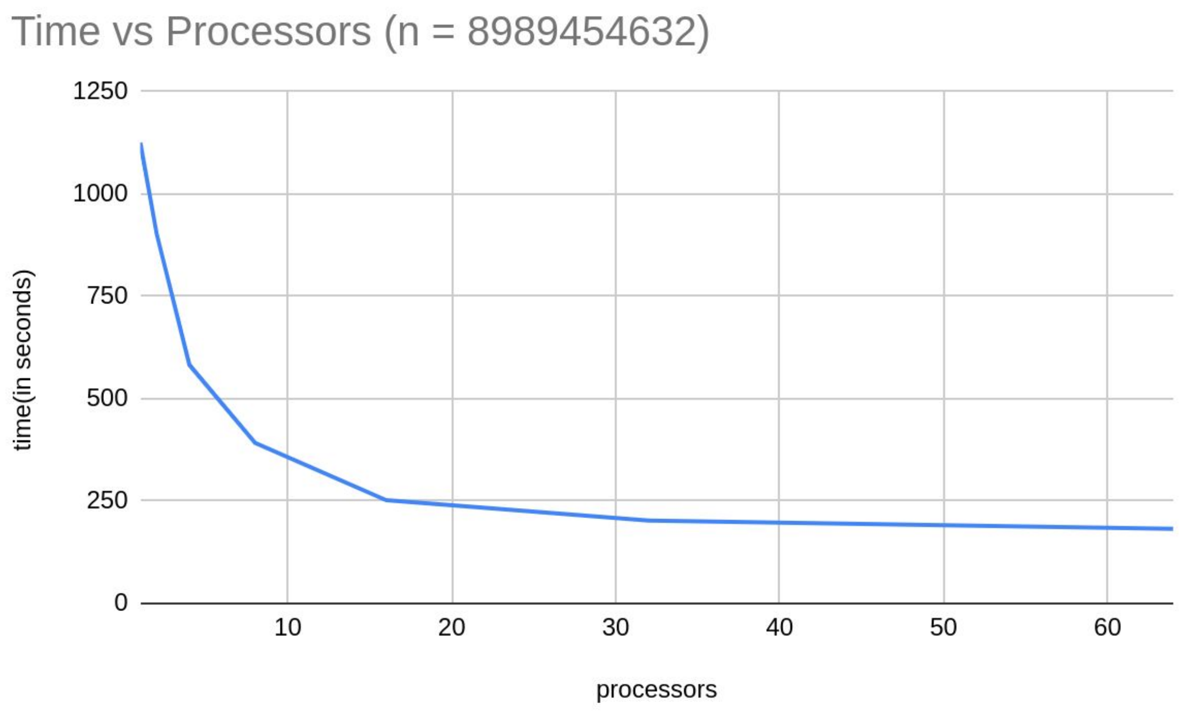


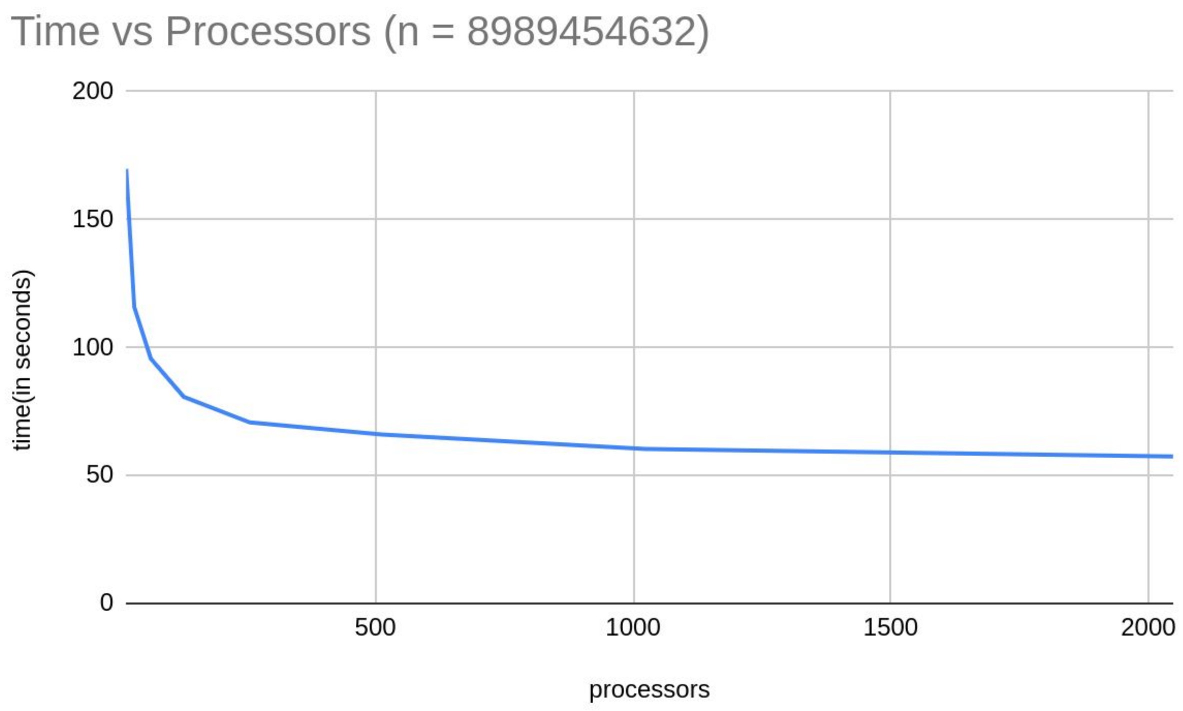
Variation for n



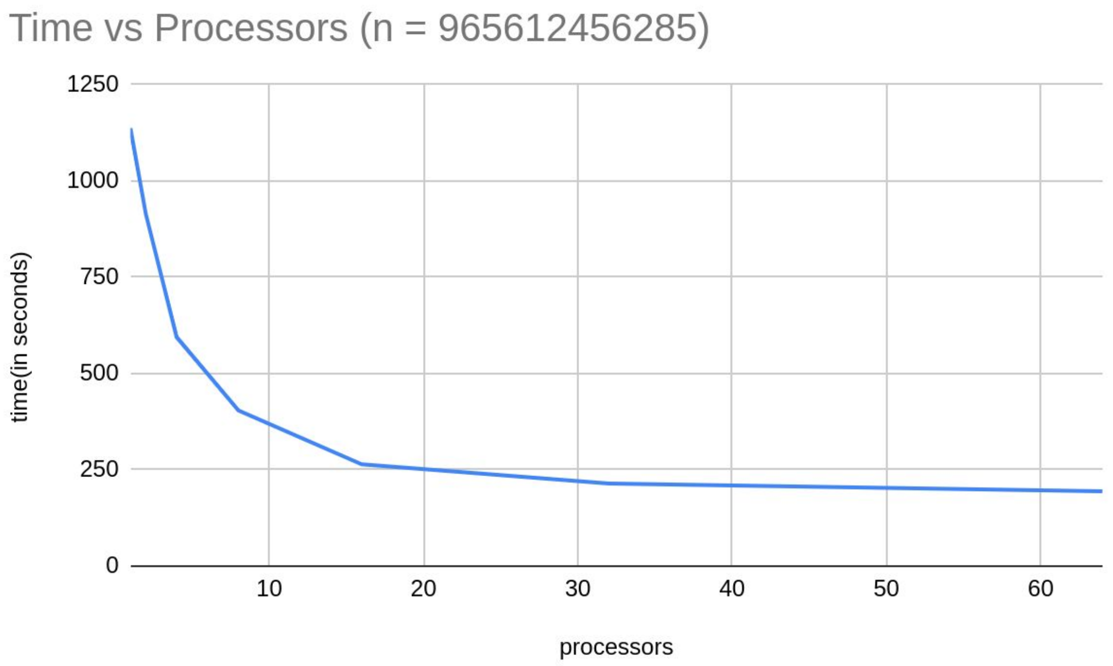


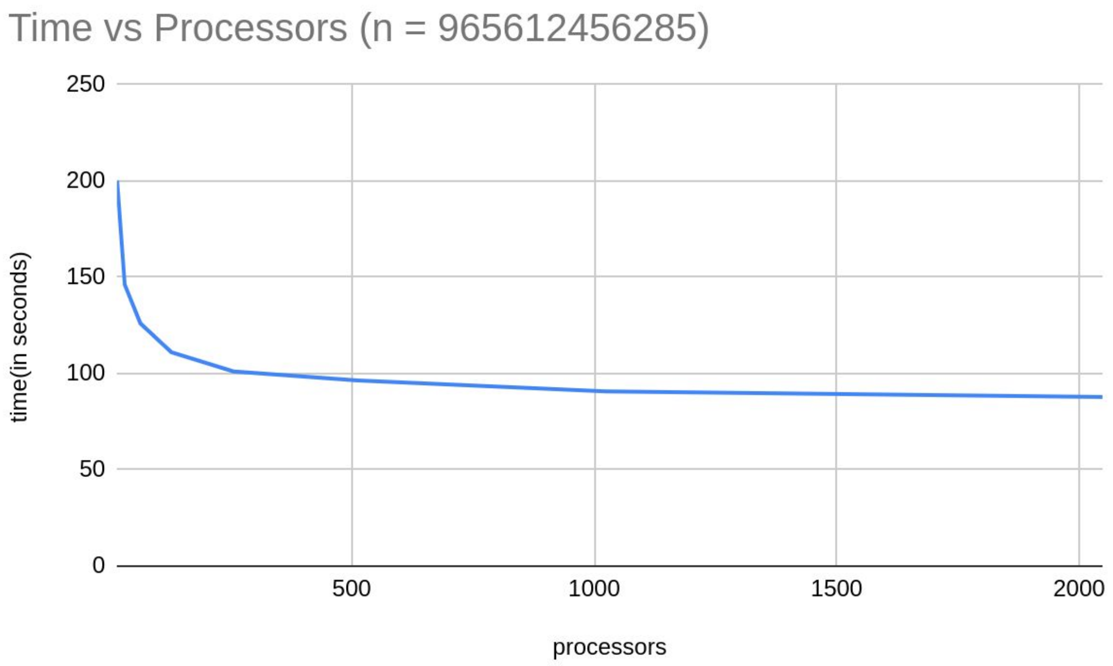
Variation for n





Variation for n



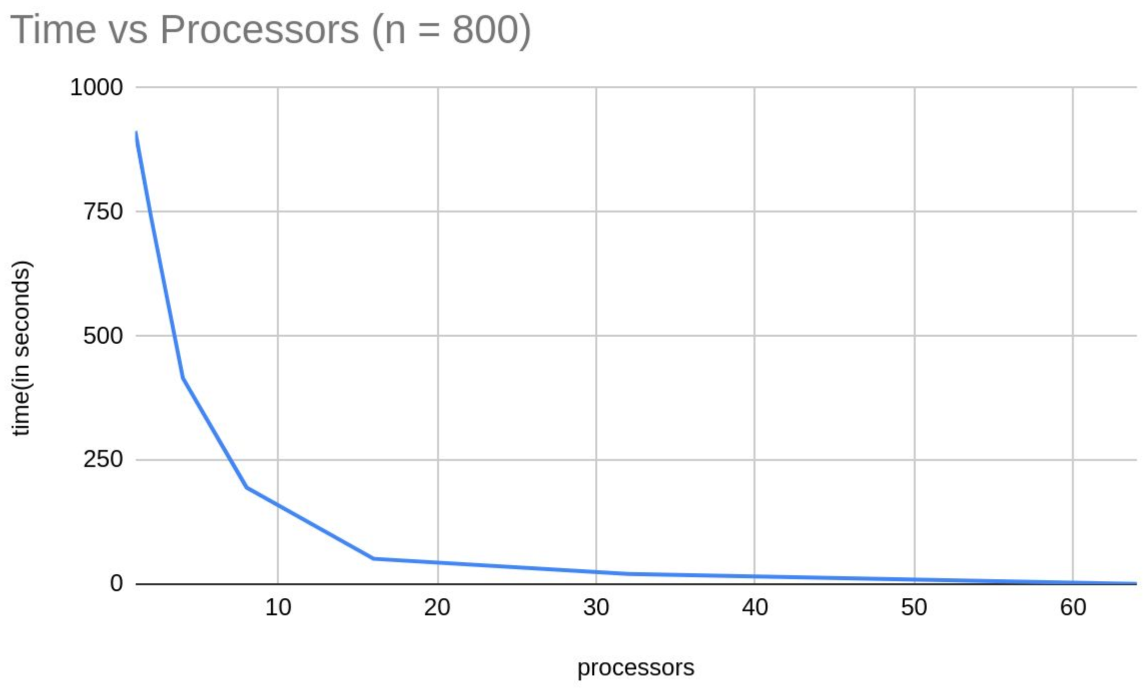


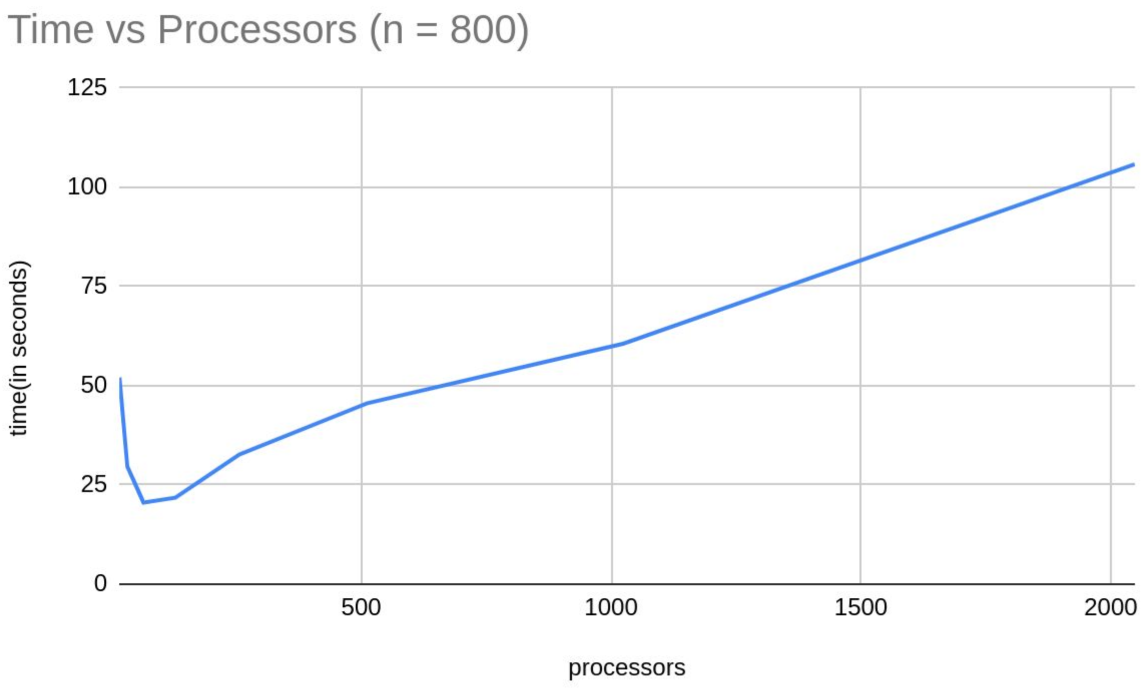
Results for small n

Processors	Time(in seconds)
1	913.254
2	739.414
4	415.516
8	194.501
16	51.047
32	20.499
64	0.148

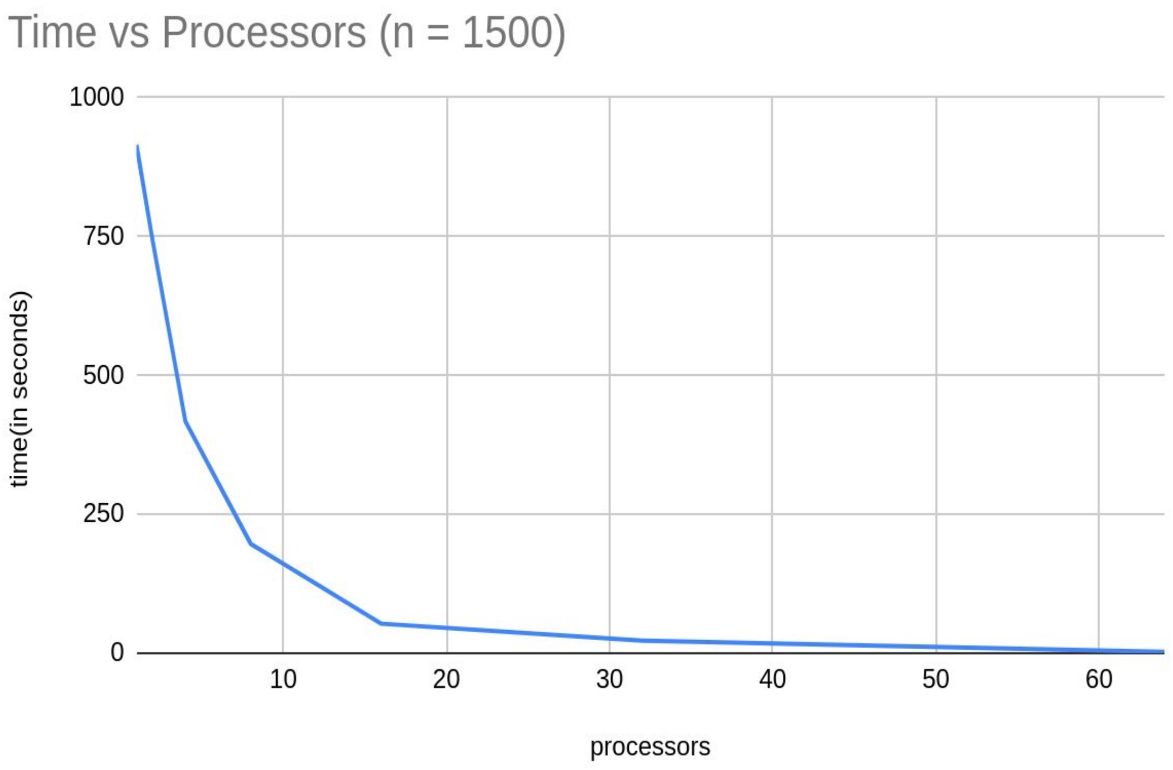
Processors	Time(in seconds)
16	52.047
32	29.565
48	25.111
64	20.569
128	21.789
256	32.645
512	45.598
1024	60.565

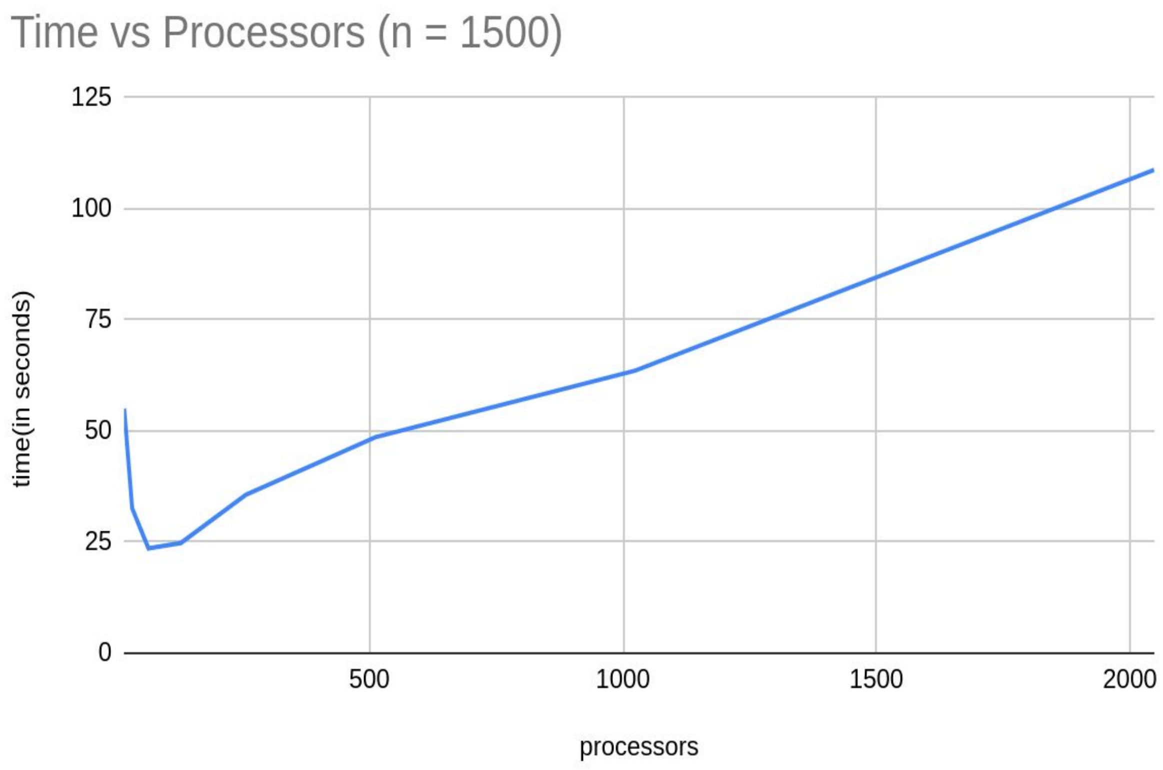
Small n


Time vs Processors (n = 800)
1000
750
500
250
0
time(in seconds)
10
20
30
40
50
60
processors


Time vs Processors (n = 800)
125
100
75
50
25
0
time(in seconds)
500
1000
1500
2000
processors

Small n


Time vs Processors (n = 1500)
1000
750
500
250
0
10
20
30
40
50
60
processors
time(in seconds)


Time vs Processors (n = 1500)
125
100
75
50
25
0
500
1000
1500
2000
processors
time(in seconds)

REFERENCES

- https://crypto.stanford.edu/cs359c/17sp/projects/JacquelineSpeiser.pdf
- https://medium.com/coinmonks/integer-factorization-defining-the-limits-of-rsa-cracking-71fc0675bc0e
- https://www.wccusd.net/cms/lib/CA01001466/Centricity/Domain/60/The%20Sieve%20of%20Eratosthenes.pdf

Thank You!
Questions?