PRIME NUMBER GENERATION

SAYLI NADKAR

sayliume@buffalo.edu

Guided by : Dr. Russ Miller

University at Buffalo The State University of New York

OUTLINE

- BACKGROUND
- SIEVE OF ERATOSTHENES ALGORITHM
- SEQUENTIAL IMPLEMENTATION
- PARALLEL IMPLEMENTATION
- RESULTS

BACKGROUND

Prime numbers are important for many reasons, both in mathematics and in practical applications :

- Cryptography
- Number theory
- Computing
- Real-world applications such as design of barcodes and numbers

SIEVE OF ERATOSTHENES

- A classic algorithm for finding the prime numbers below a certain integer (limit)
- Marks out composite numbers - the multiples of each prime starting with 2 (or 3).
- It has a time complexity of O(n log log n)
- The Sieve of Eratosthenes is still used today in many applications, such as cryptography, number theory, and computer science.

	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100

WORKING - Sieve of Eratosthenes

HOW DOES THE ALGORITHM WORK?

```
algorithm Sieve of Eratosthenes is
    input: an integer n > 1.
    output: all prime numbers from 2 through n.

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

    return all i such that A[i] is true.
```

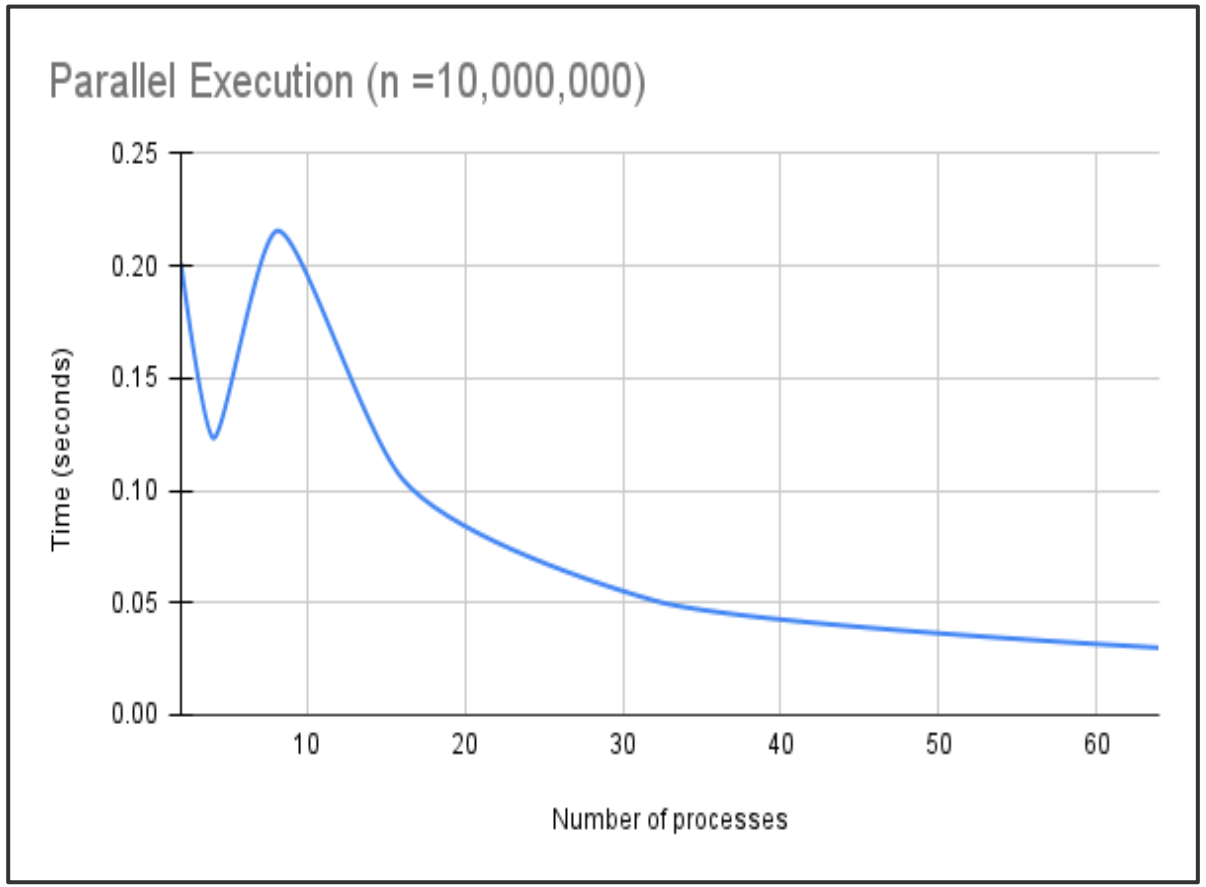
NEED FOR PARALLELIZATION

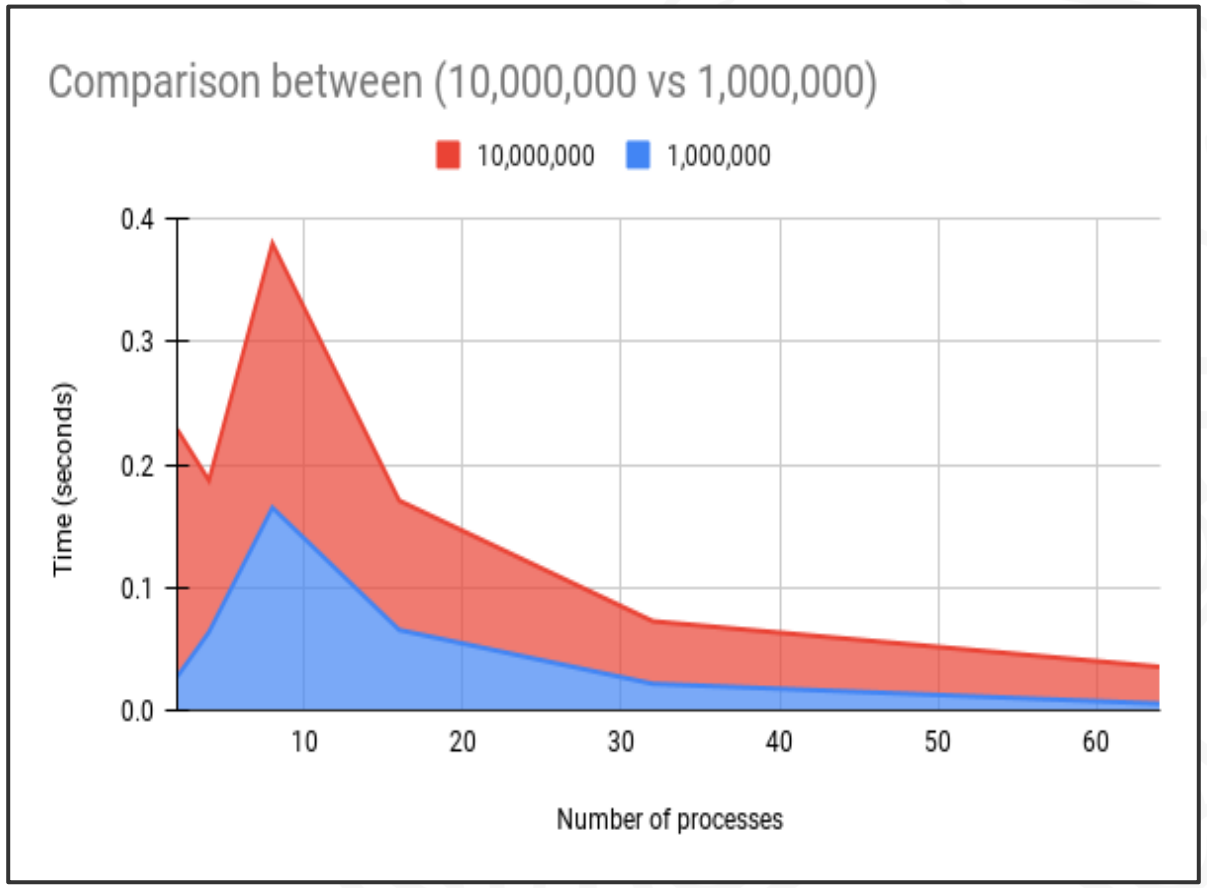
- Improve Performance
- Faster Execution time
- Scale efficiently to larger input sizes

PARALLELIZATION STRATEGY

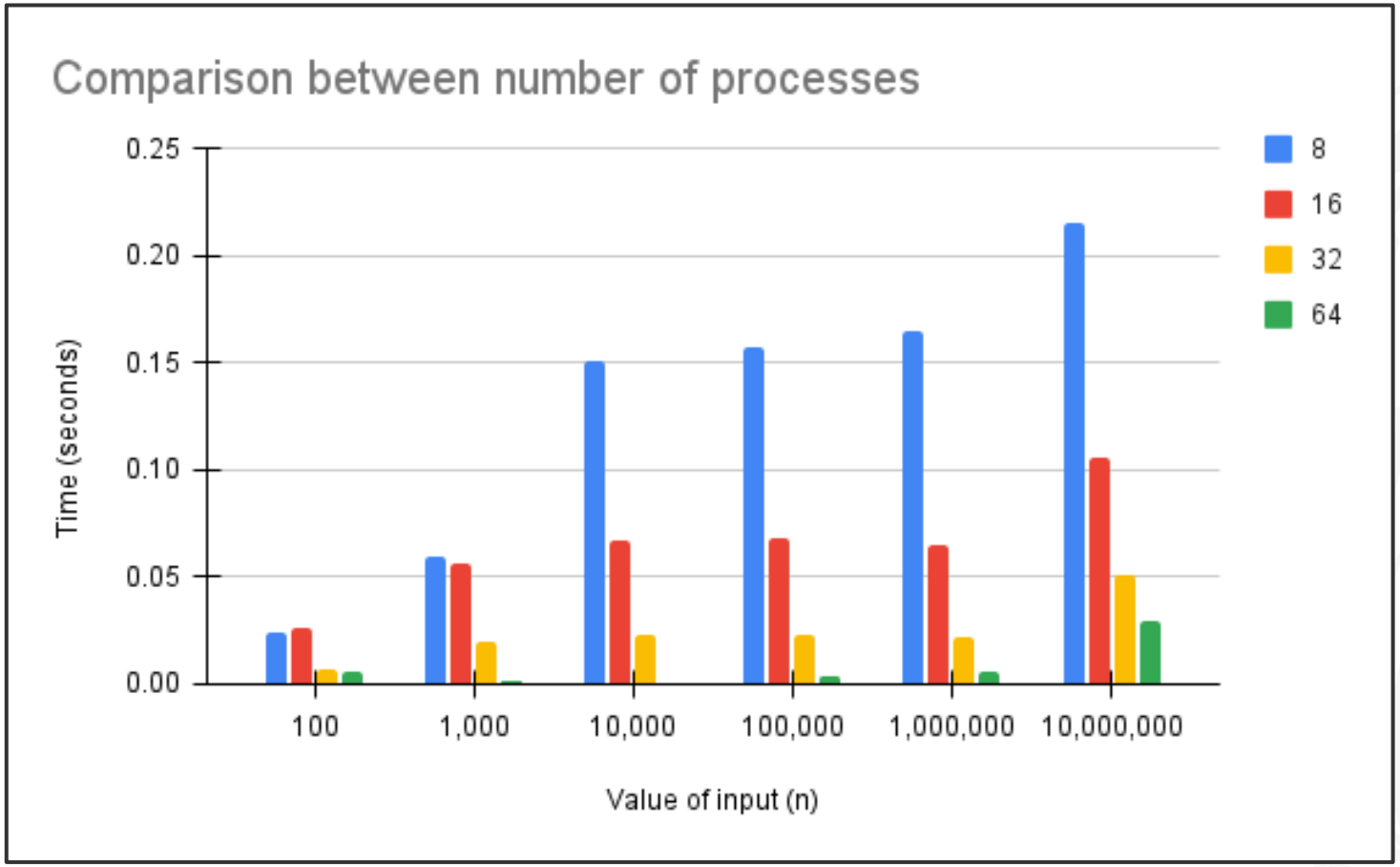
- Distributing the input range across multiple machines
- Machine processes a subset of the range independently
- Communicates with other machines to ensure each prime number is marked only once.

RESULTS


Parallel Execution (n =10,000,000)
0.25
0.20
0.15
0.10
0.05
0.00
Time (seconds)
10
20
30
40
50
60
Number of processes


Comparison between (10,000,000 vs 1,000,000)
10,000,000
1,000,000
0.4
0.3
0.2
0.1
0.0
Time (seconds)
10
20
30
40
50
60
Number of processes

RESULTS


Comparison between number of processes
0.25
0.20
0.15
0.10
0.05
0.00
Time (seconds)
8
16
32
64
100
1,000
10,000
100,000
1,000,000
10,000,000
Value of input (n)

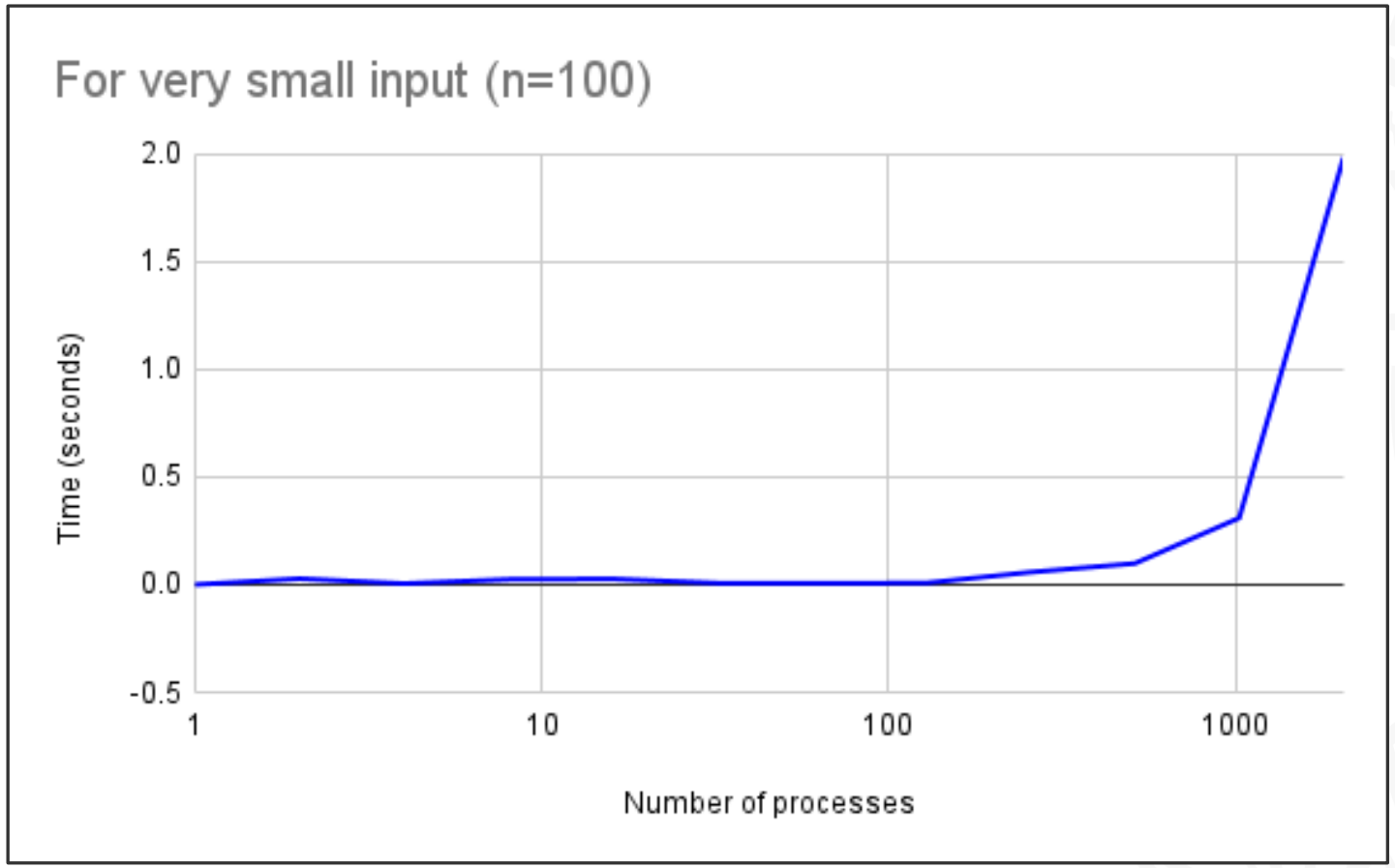
FUTURE WORK

- Scaling up the number of processes. ✓
- Re- run for a larger input limit. ✓

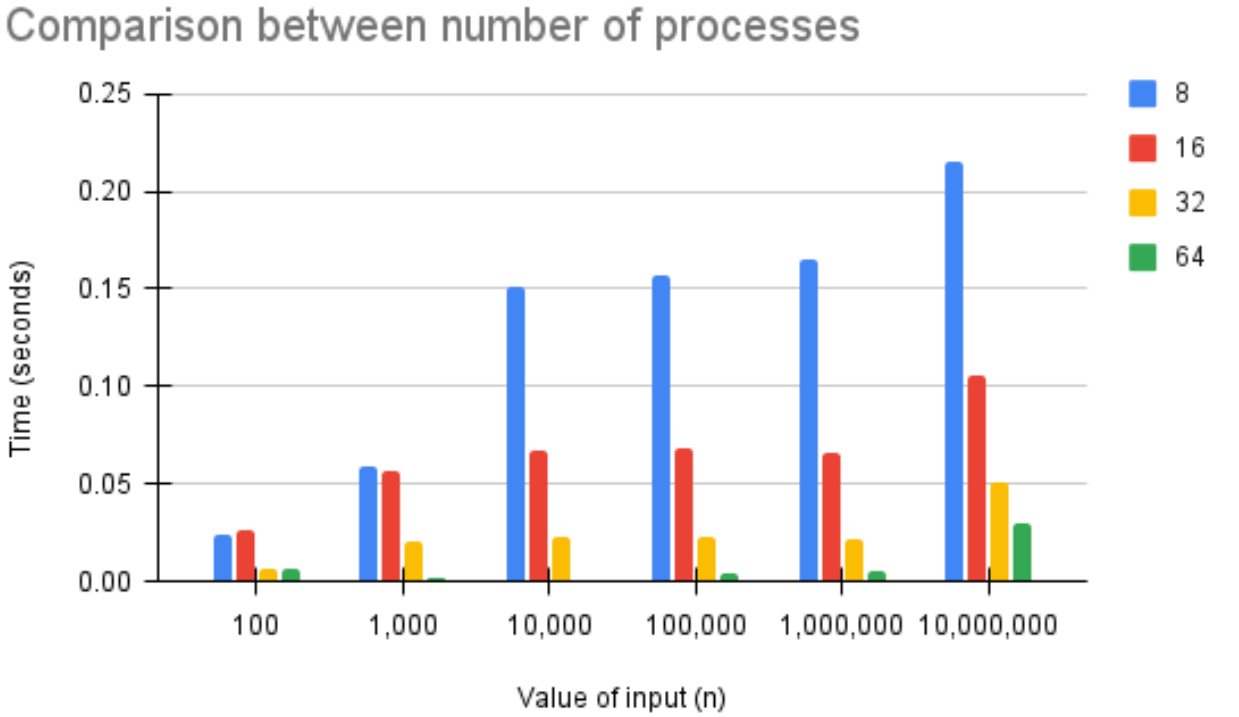
Results :

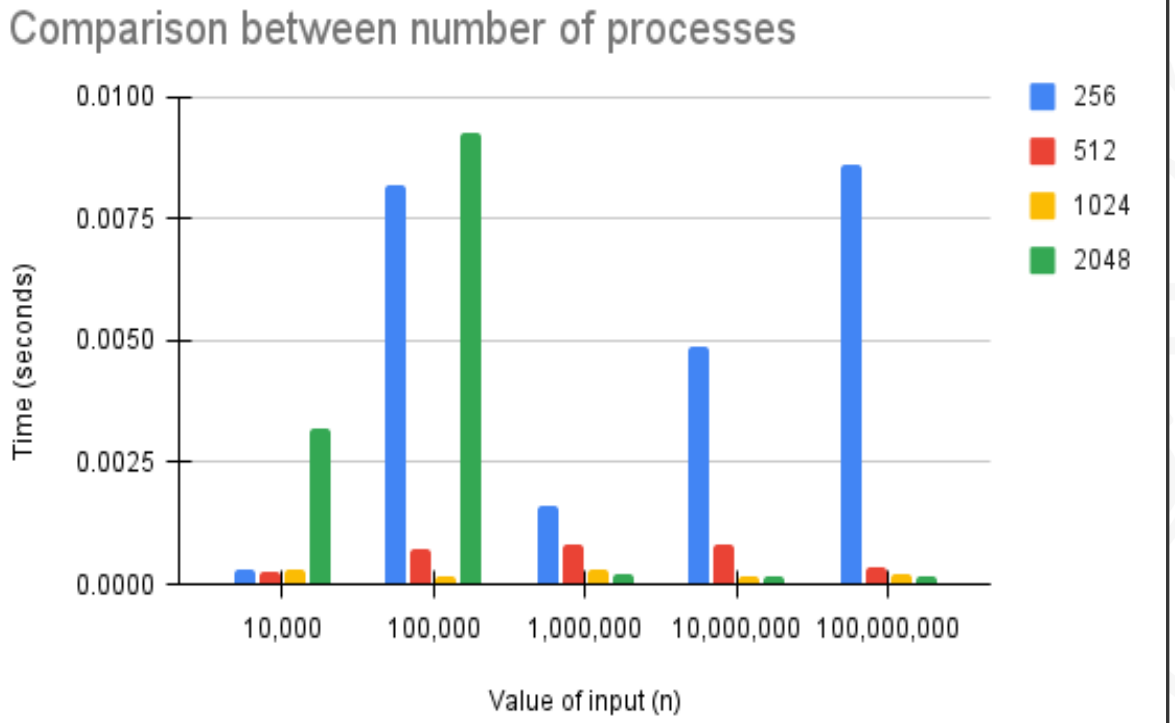
Number of processes	100	1000	10000	100000	1000000	10000000	100000000
1	0.000086	0.000104	0.000228	0.001516	0.019127	0.219986	0.69826
2 (n=2, p=1)	0.027407	0.015359	0.023652	0.016511	0.027383	0.201297	0.47937
4 (n=4 p=1)	0.004686	0.041508	0.071409	0.064771	0.063643	0.12356	0.2483
8 (n=8, p=1)	0.024056	0.059248	0.150896	0.156914	0.165036	0.215375	0.23874
16 (n=8, p=2)	0.026213	0.056028	0.066826	0.067866	0.065394	0.105376	0.19266
32 (n=8, p=4)	0.006969	0.020136	0.023114	0.022663	0.02158	0.050924	0.094875
64 (n=8, p=8)	0.006386	0.001665	0.000499	0.004217	0.00539	0.029957	0.04819
128(n=8, p=16)	0.007428	0.000575	0.000404	0.002986	0.003134	0.0088	0.014926
256(n=16, p=16)	0.0576	0.000485	0.000294	0.008167	0.00163	0.00486	0.008612
512(n=32, p=16)	0.09814	0.00897	0.00025	0.0007243	0.00082	0.000838	0.0003384
1024(n=64, p=16)	0.3093	0.00129	0.000319	0.0001724	0.0002893	0.000185	0.0002194
2048(n=128, p=16)	1.983	0.08673	0.00321	0.00928	0.0001956	0.000149	0.0001605

RESULTS

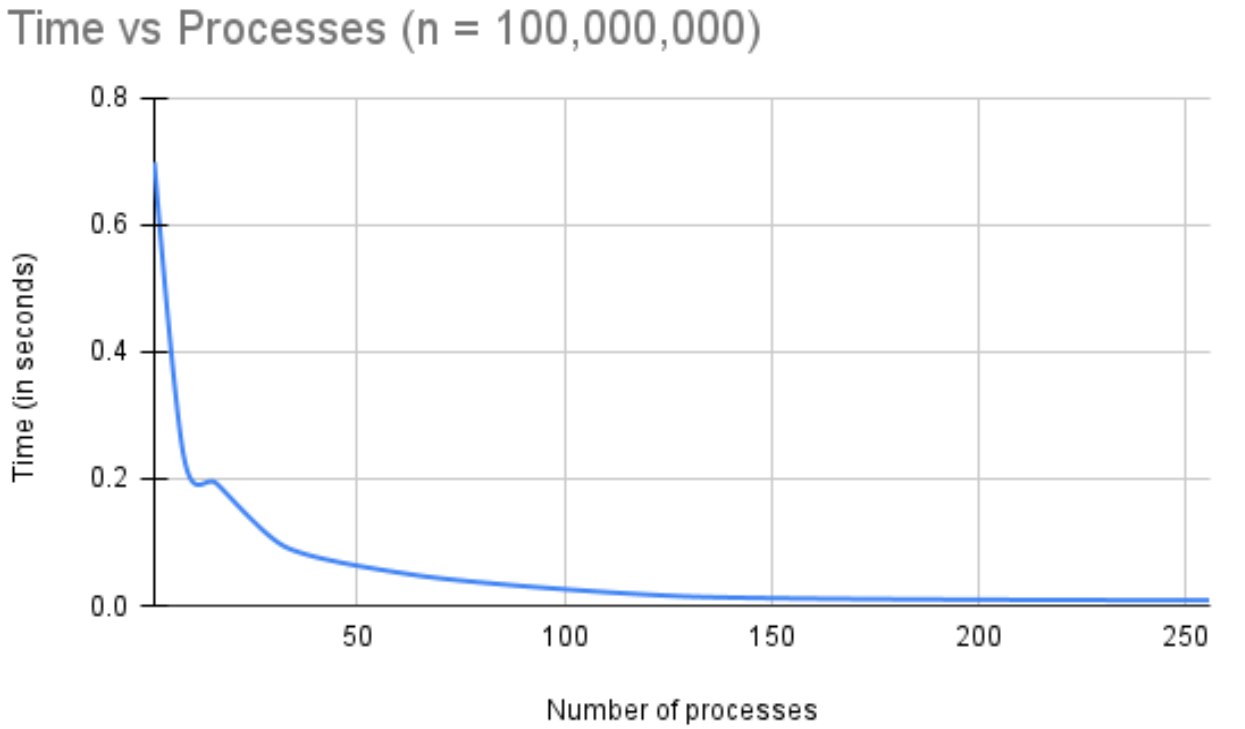

For very small input (n=100)
2.0
1.5
1.0
0.5
0.0
-0.5
Time (seconds)
1
10
100
1000
Number of processes

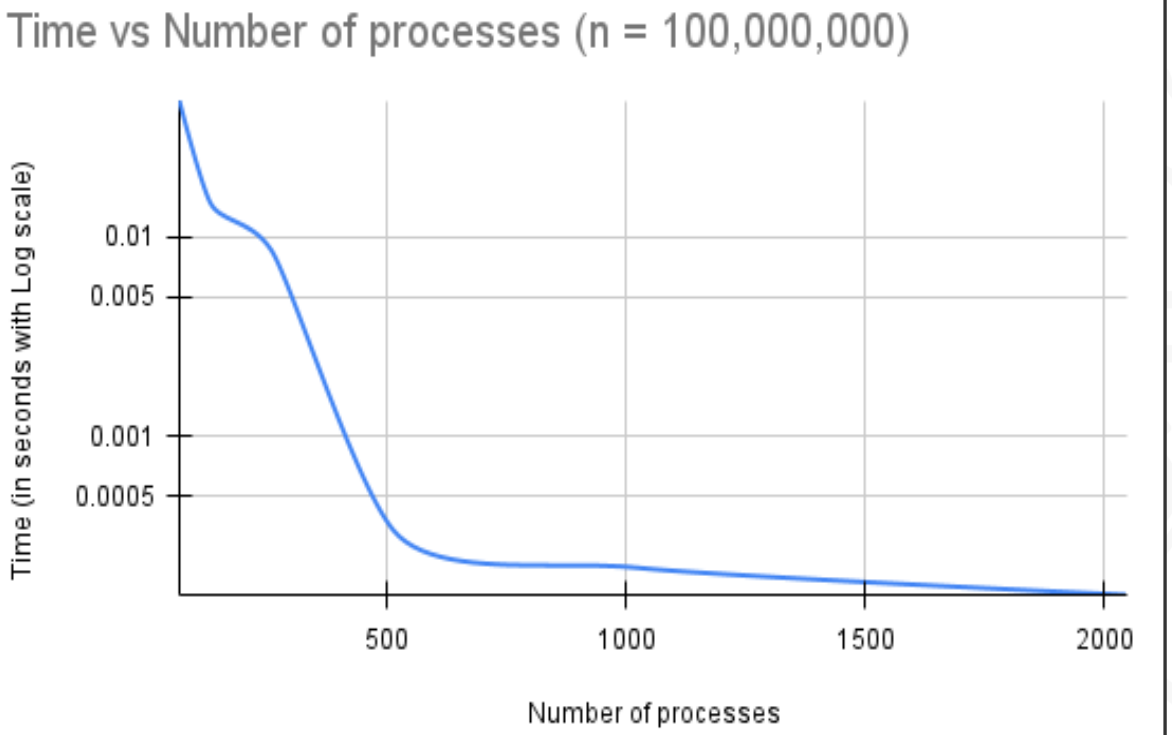
RESULTS


Comparison between number of processes
0.25
0.20
0.15
0.10
0.05
0.00
Time (seconds)
100
1,000
10,000
100,000
1,000,000
10,000,000
Value of input (n)
8
16
32
64

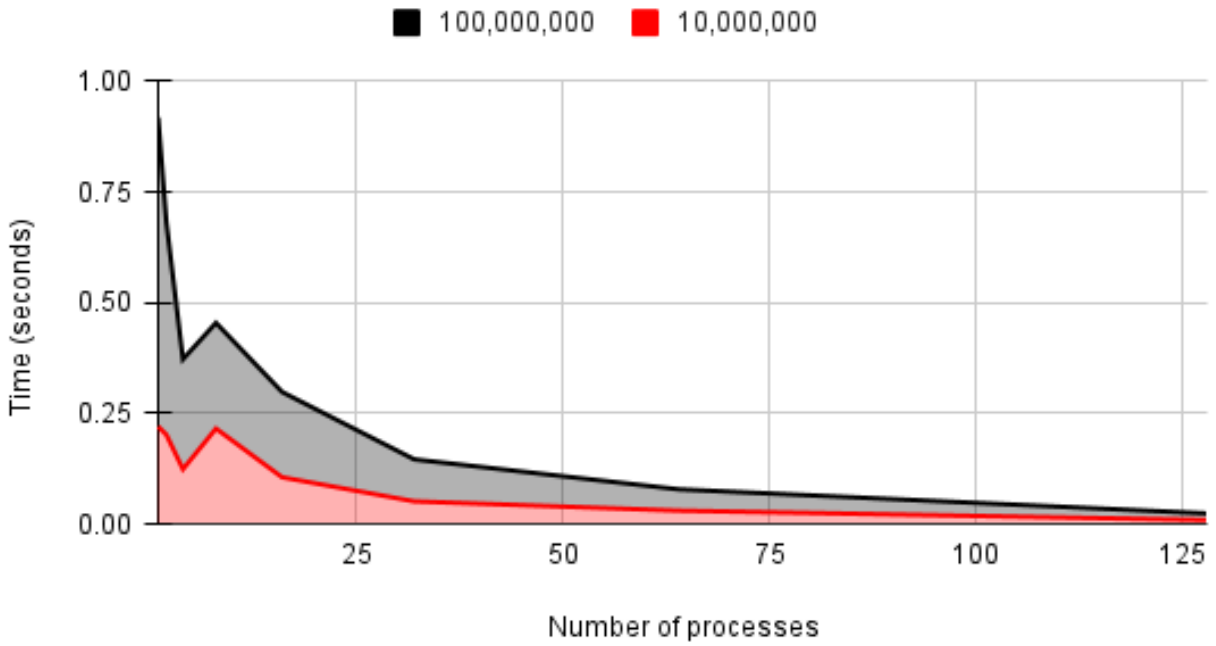

Comparison between number of processes
0.0100
0.0075
0.0050
0.0025
0.0000
Time (seconds)
10,000
100,000
1,000,000
10,000,000
100,000,000
Value of input (n)
256
512
1024
2048

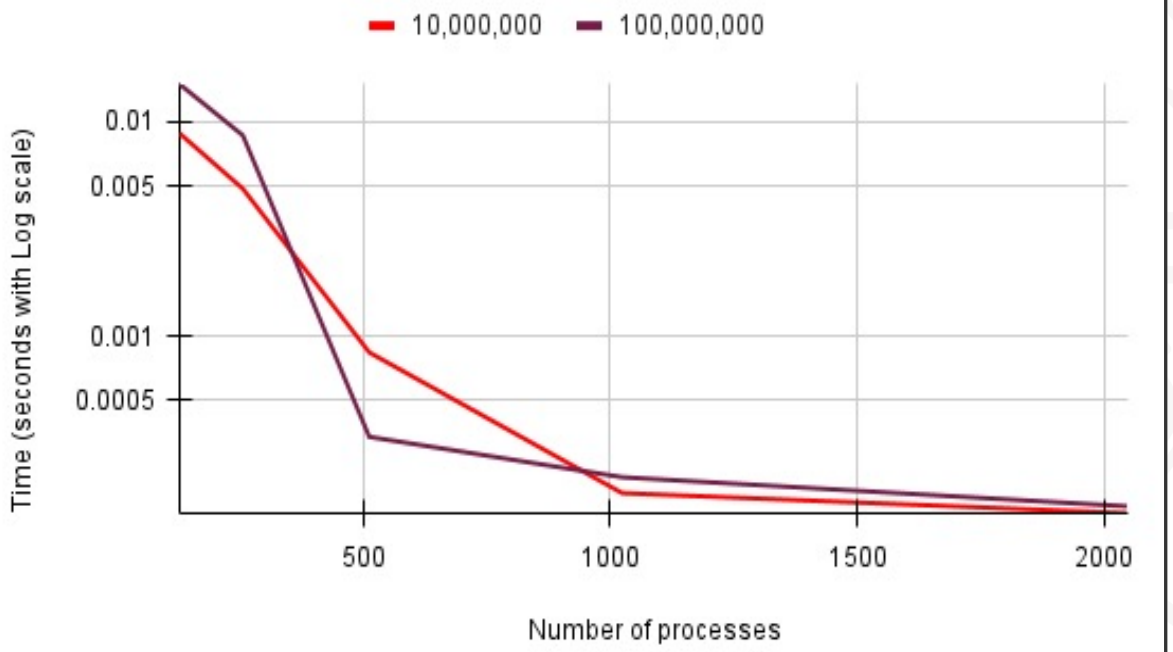
RESULTS


Time vs Processes (n = 100,000,000)
Time (in seconds)
0.8
0.6
0.4
0.2
0.0
50
100
150
200
250
Number of processes

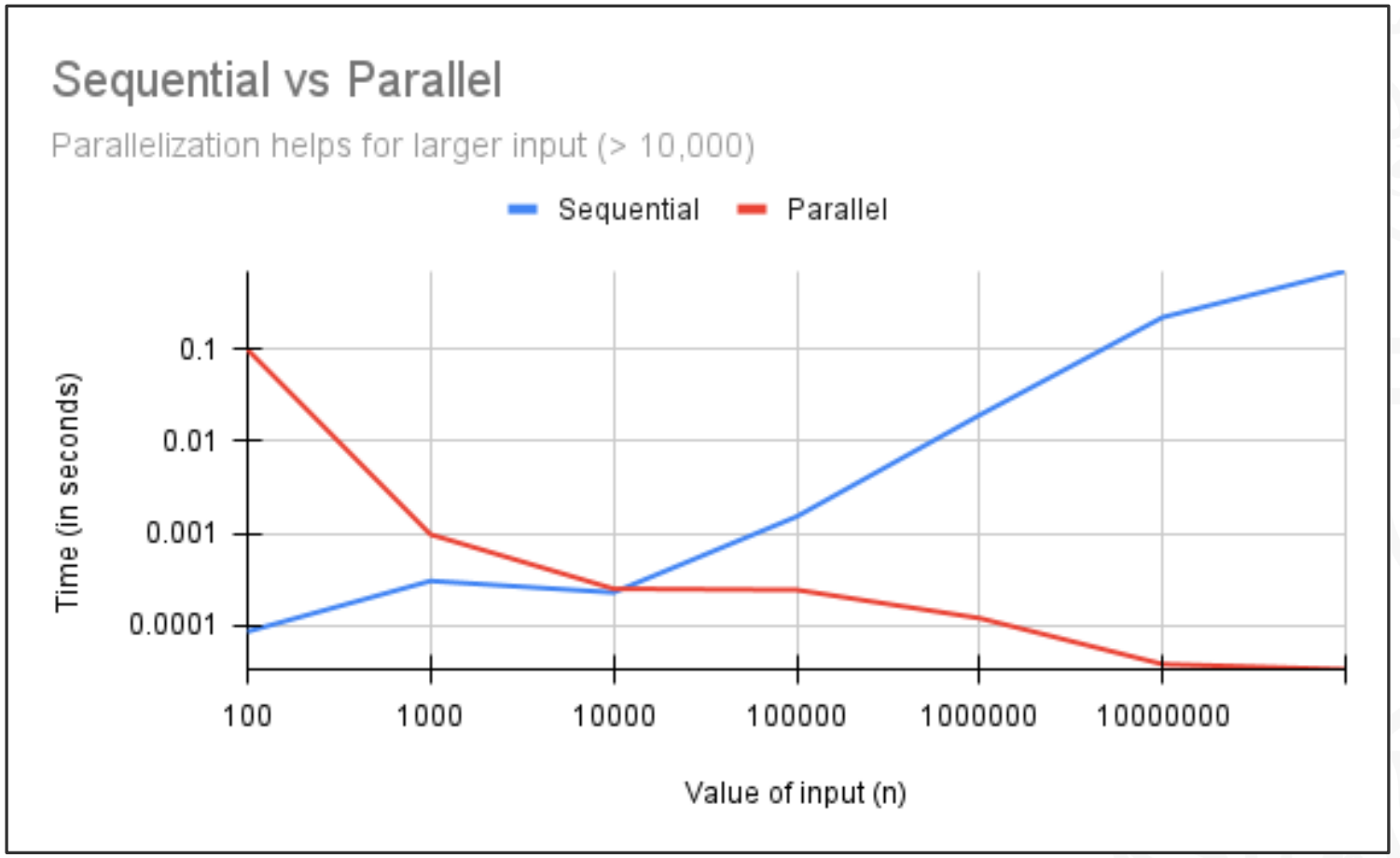

Time vs Number of processes (n = 100,000,000)
Time (in seconds with Log scale)
0.01
0.005
0.001
0.0005
500
1000
1500
2000
Number of processes

RESULTS


Comparison between input values
100,000,000
10,000,000
1.00
0.75
0.50
0.25
0.00
Time (seconds)
25
50
75
100
125
Number of processes


Comparison between input values
10,000,000
100,000,000
0.01
0.005
0.001
0.0005
Time (seconds with Log scale)
500
1000
1500
2000
Number of processes

RESULTS


Sequential vs Parallel
Parallelization helps for larger input (> 10,000)
Sequential
Parallel
Time (in seconds)
0.1
0.01
0.001
0.0001
100
1000
10000
100000
1000000
10000000
Value of input (n)

REFERENCES

- https://en.wikipedia.org/wiki/Sieve_of_Eratosthenes - Euler's_sieve
- https://ubccr.freshdesk.com/support/solutions/articles/13000026245-tutorials-workshops-and-training-documents
- https://docs.ccr.buffalo.edu/en/latest/
- Hwang, S., Chung, K., Kim, D. Load Balanced Parallel Prime Number Generator with Sieve. Of Eratosthenes on Cluster Computers. IEEE 2007
- Bhat, Ganapathi & Kini, N. & Ray, Siddharth & Prabhu, Srikanth & Hegde, Govardhan. (2014). Parallelization of Sieve of Eratosthenes. International Journal of Scientific Research in Computer Science Applications and Management Studies. 3.

University at Buffalo The State University of New York

THANK YOU