Working with Prime Numbers

Kiran Radhakrishnan

CSE 708

Agenda

Introduction

Areas of Focus

What are we looking for

Summary

Introduction

Prime number is a natural number greater than 1 that cannot be expressed as the product of two natural numbers lesser than itself.

Introduction

We will explore the application of the Sieve of Eratosthenes algorithm in finding prime numbers between 1 and N

Our focus will be on analyzing the algorithm's performance when parallelized using the Message Passing Interface (MPI)

I think prime numbers are like life. They are very logical, but you could never work out the rules, even if you spent all your time thinking about them

Mark Haddon

Areas of focus

Prime factorization

Finding all prime factors of numbers from 0 to 4294967295

Improving the speed by running this parallelly

Sieve of Eratosthenes

Algorithm Overview

- The Sieve of Eratosthenes is an ancient algorithm for finding prime numbers within a specified range.
- It systematically eliminates composite numbers, leaving behind the prime numbers.
- For example, to find primes up to 30, we mark multiples of each prime starting from 2.

Pseudocode - Sequential

Algorithm Overview:

O(n log log n)

1. Create a list of Boolean values, with indices from 2 to the desired limit, initially all set to true.

2. Set a variable "p" to 2, the first prime number.

3. Repeat the following steps until p * p is less than or equal to the limit:

    a. If the value at index p is true (i.e., p is not marked as composite): i. Mark all multiples of p (excluding p itself) as composite by setting their values to false.

    b. Find the next unmarked number greater than p, and set it as the new value of p. This is the next prime number.

4. All the unmarked numbers that are still set to true in the list are prime numbers.

Pseudocode – Parallel Part 1

Algorithm Overview:

1.Initialize MPI and process information.

2.Check for the command-line argument, 'n.’

3.Calculate local range and array size.

4.Allocate memory for the 'marked' array.

5.Initialize 'prime' on the master process.

Pseudocode – Parallel Part 2

Algorithm Overview:

6. Implement the Sieve of Eratosthenes algorithm:
    1. Mark multiples of the current 'prime' within the local range.
    2. Update 'prime' by finding the next unmarked number.
    3. Broadcast the new 'prime' to all processes.
    4. Repeat until 'prime' squared exceeds 'n.’

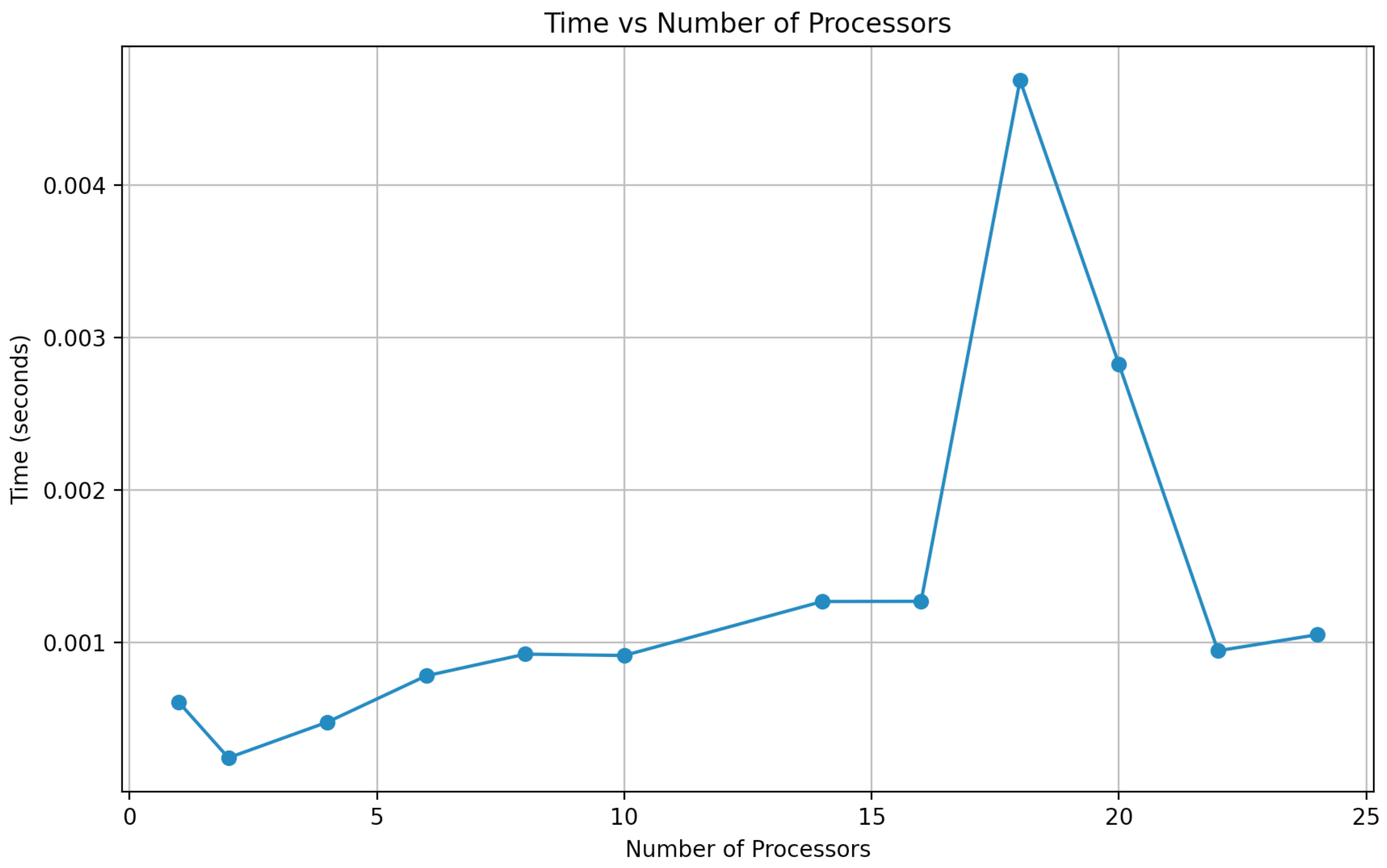
7. Count local primes and reduce to get 'global_count.’

8. Measure execution time.

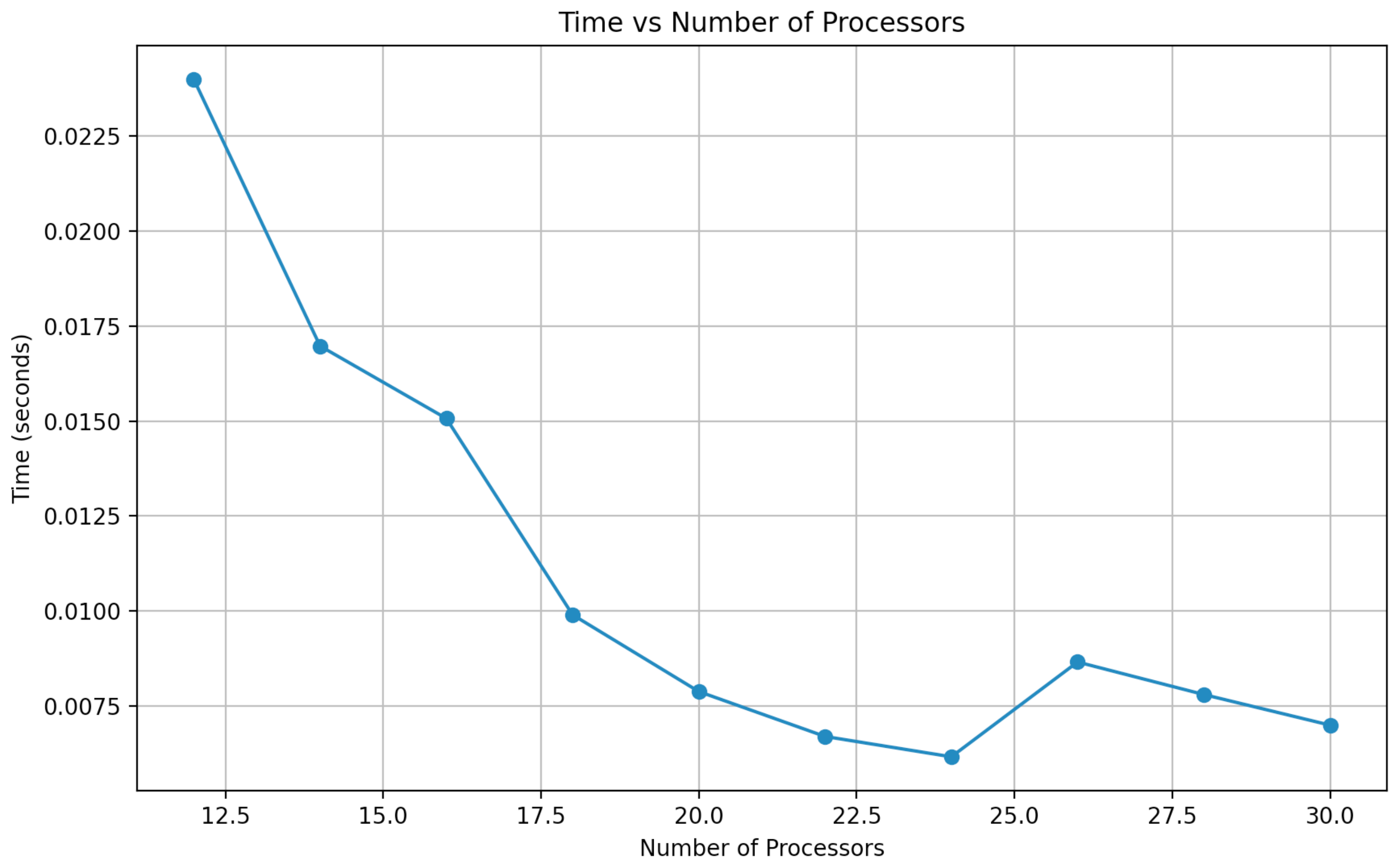
9. Print the number of processes, 'global_count,' and execution time.

10. Finalize MPI and exit.

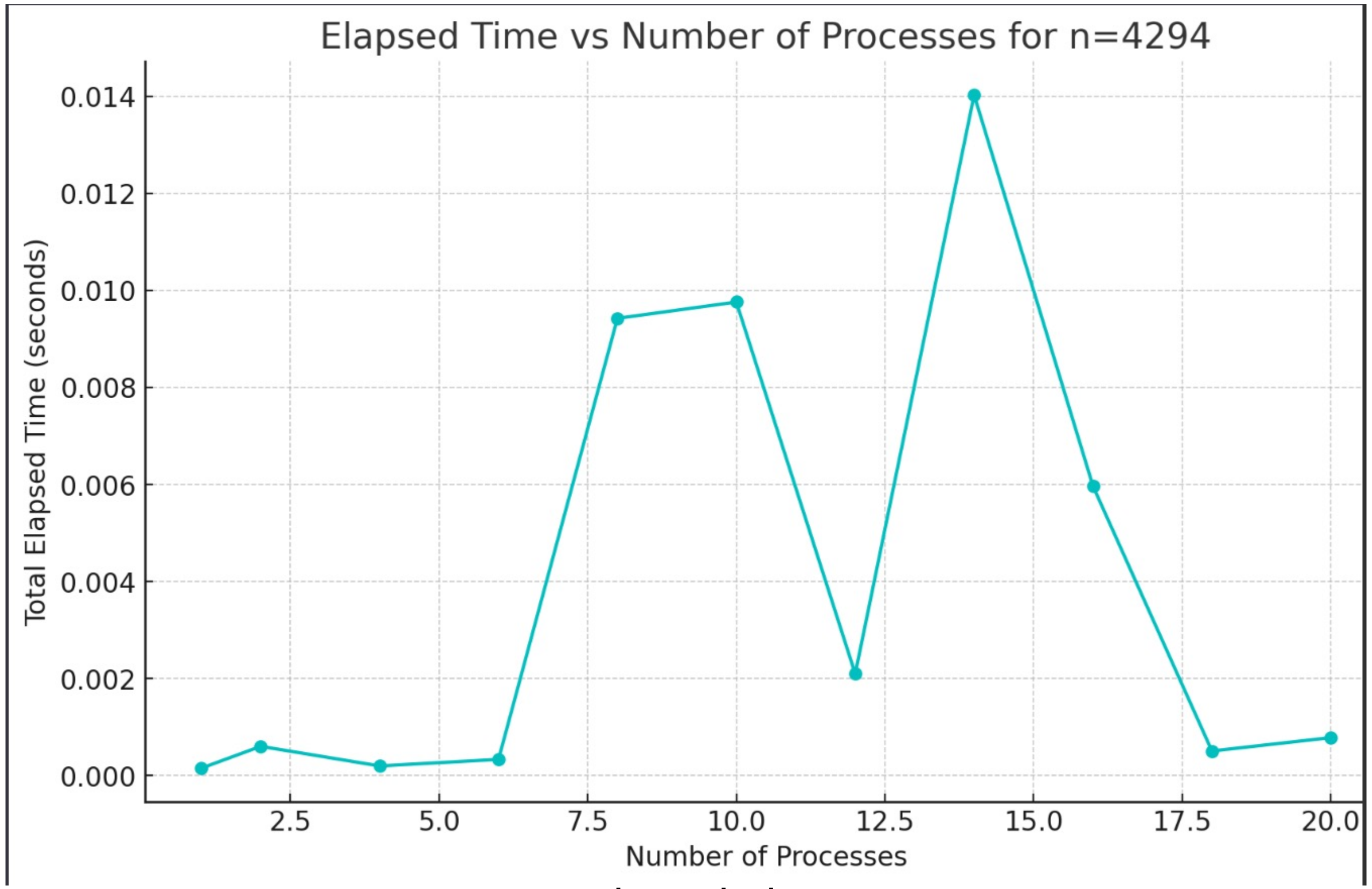
Performance graphs (n=500)



Performance graphs(n= 429496729)

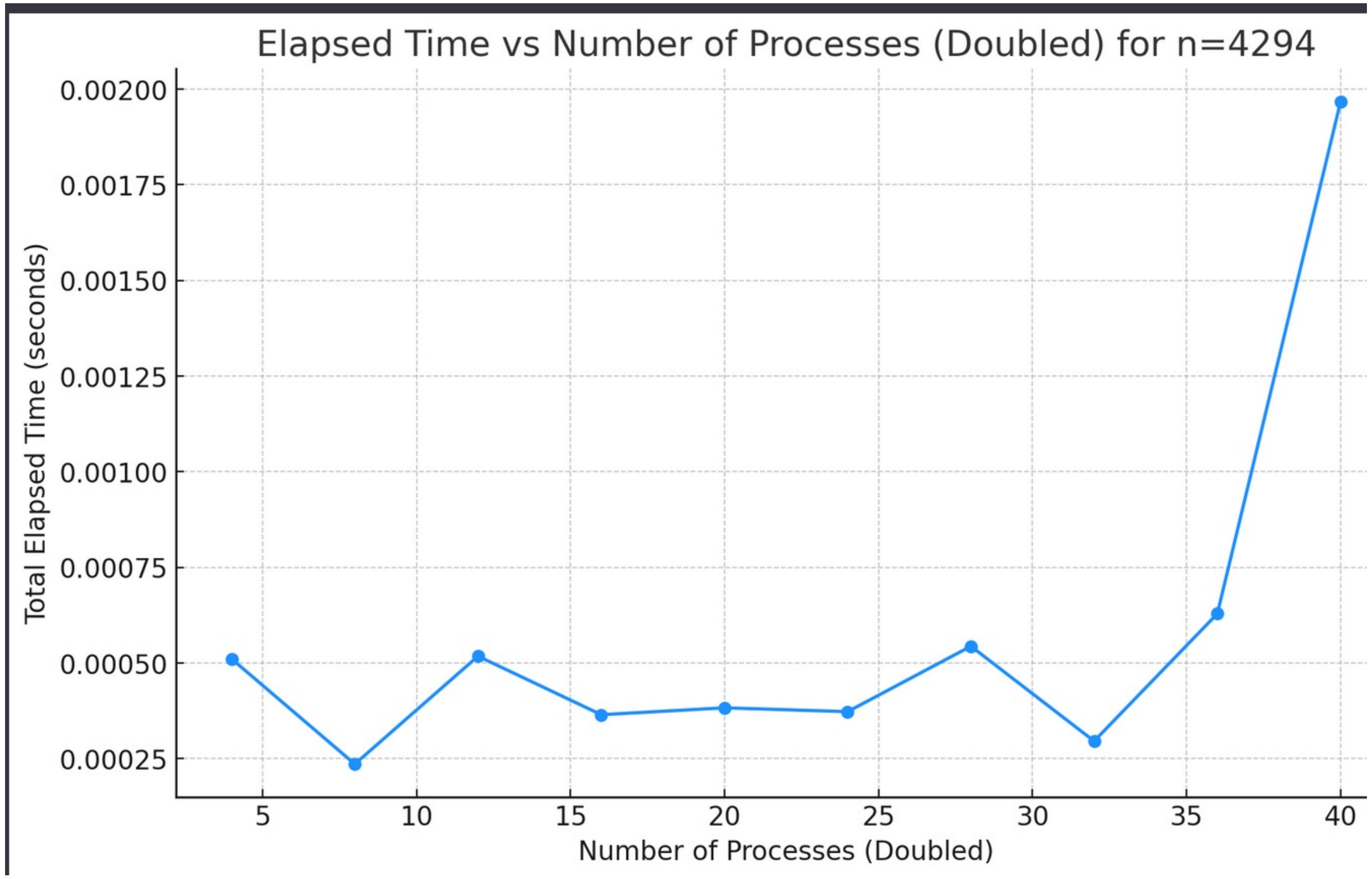


N = 4294 (1 task per node)

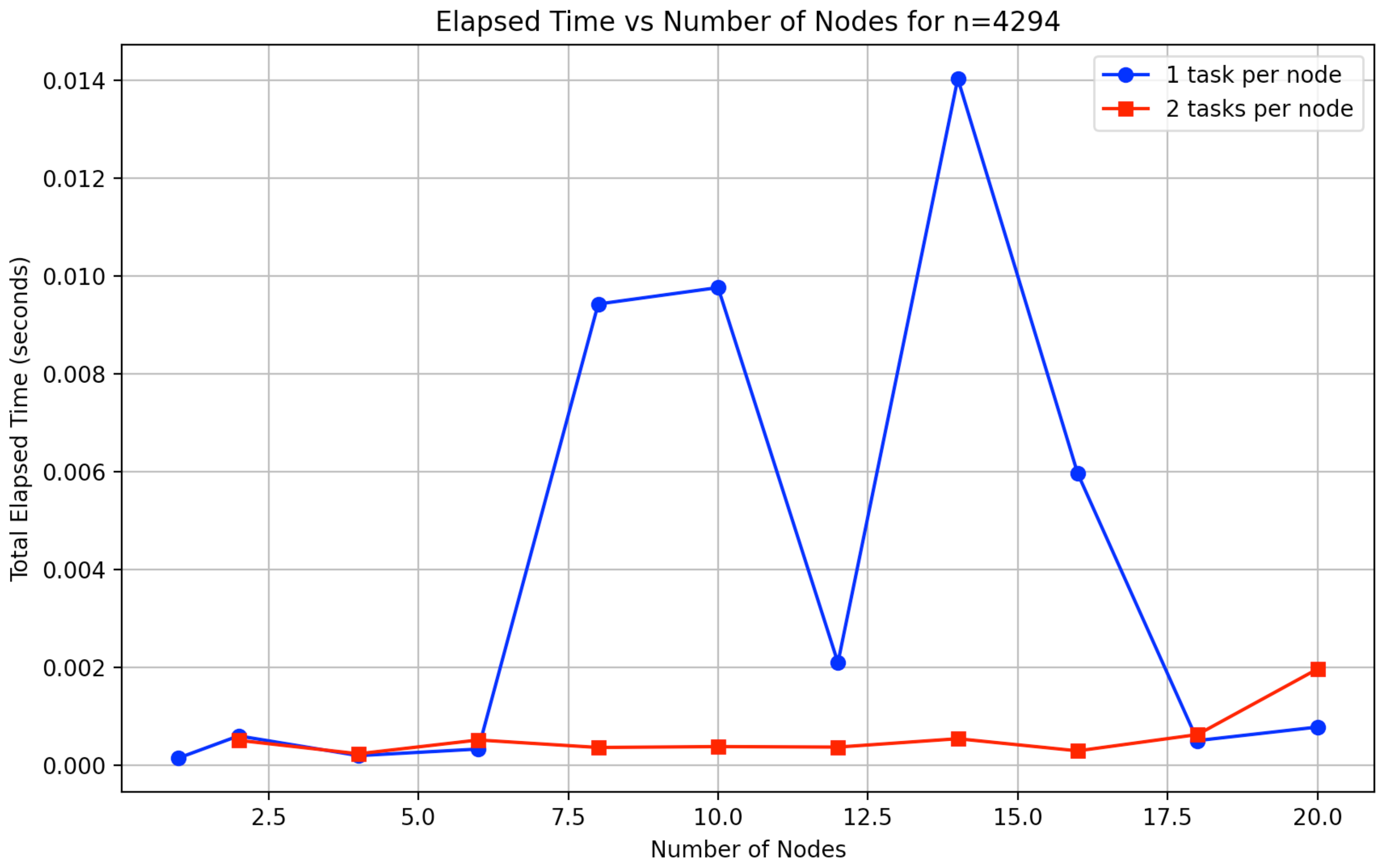


Each node has 1 CPU

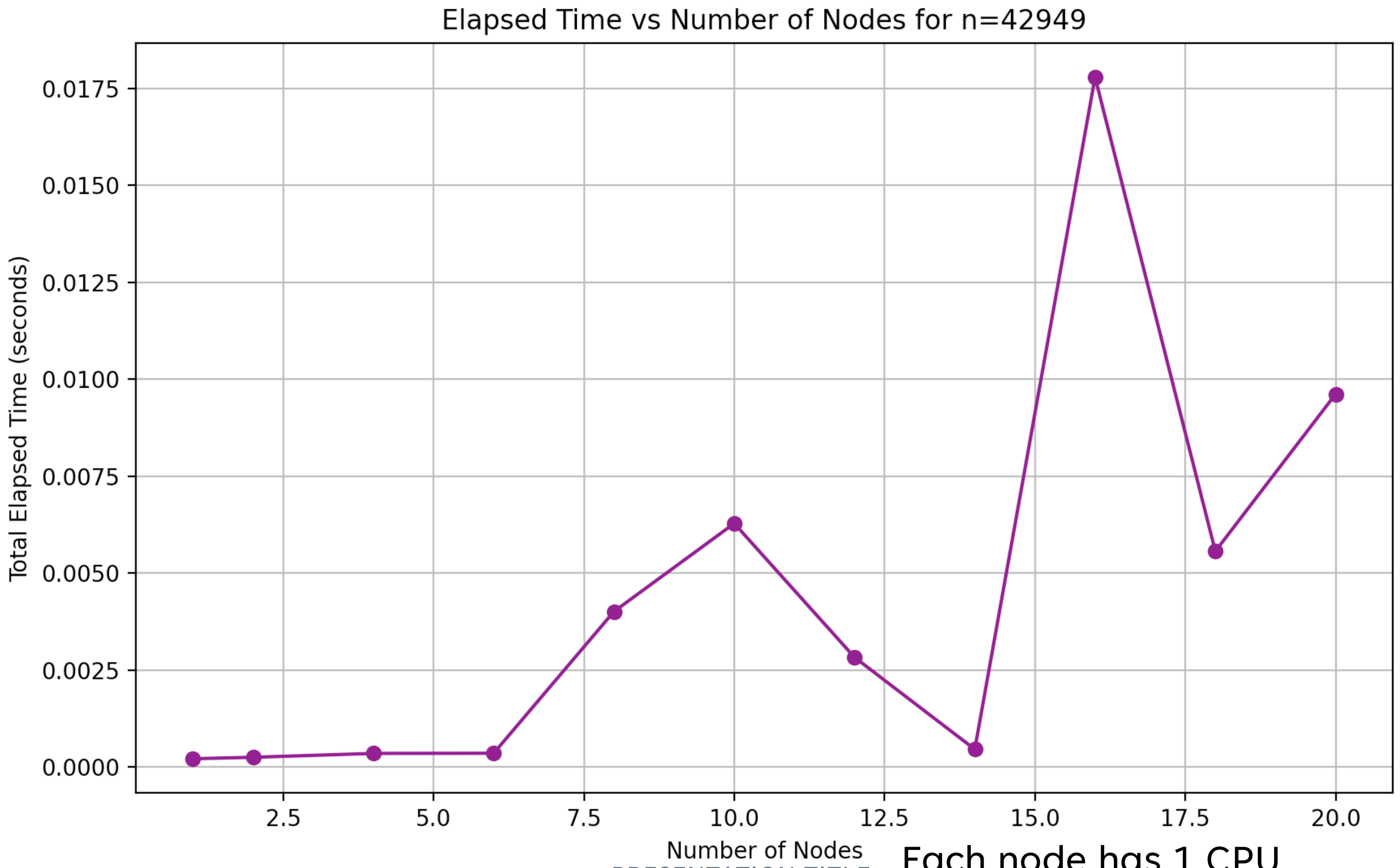
N = 4294 (2 tasks per node)



Comparing both graphs

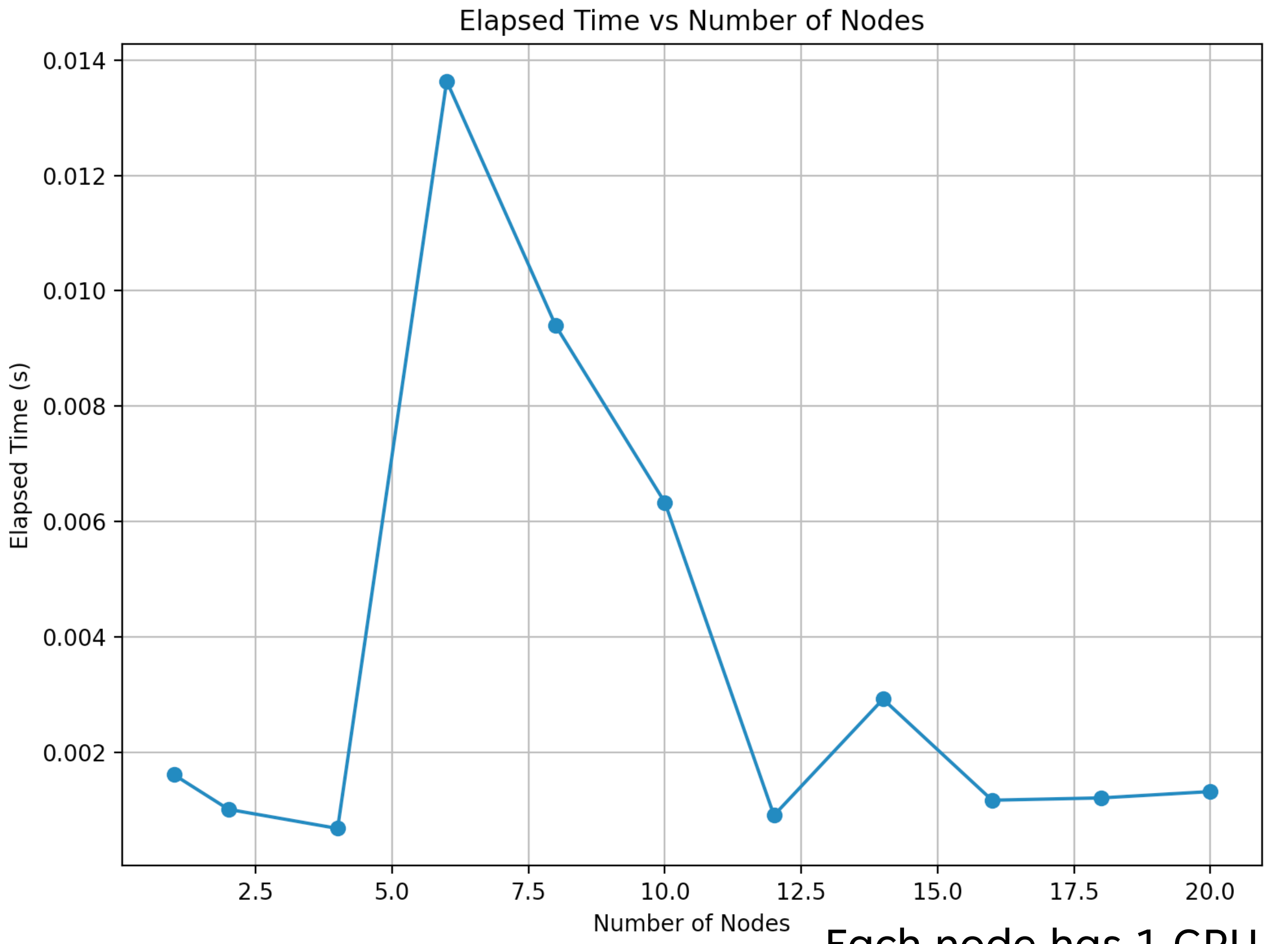


N = 42949 (1 task per node)



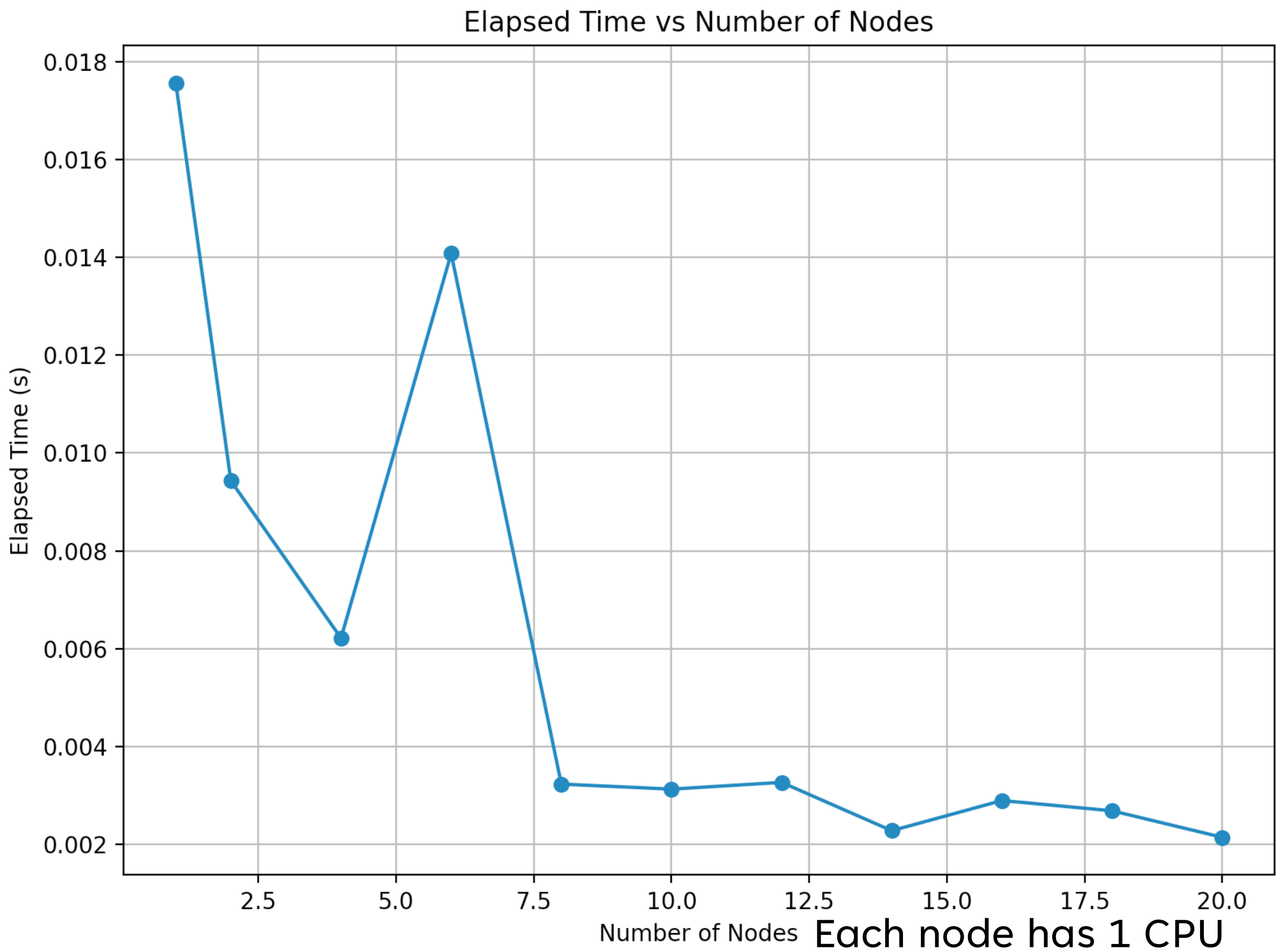
Each node has 1 CPU

N = 429496 (1 task per node)



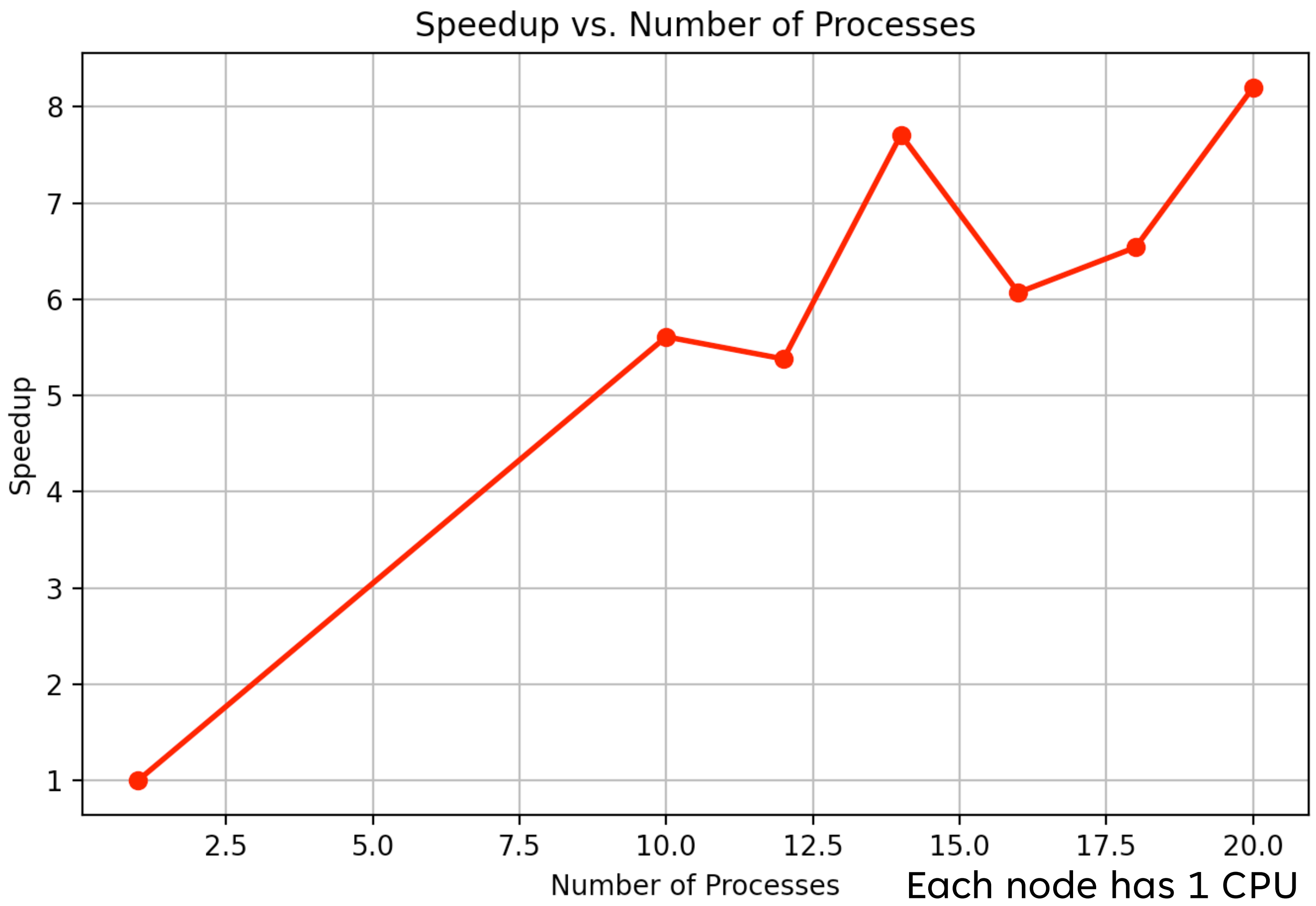
Each node has 1 CPU

N = 4294967 (1 task per node)



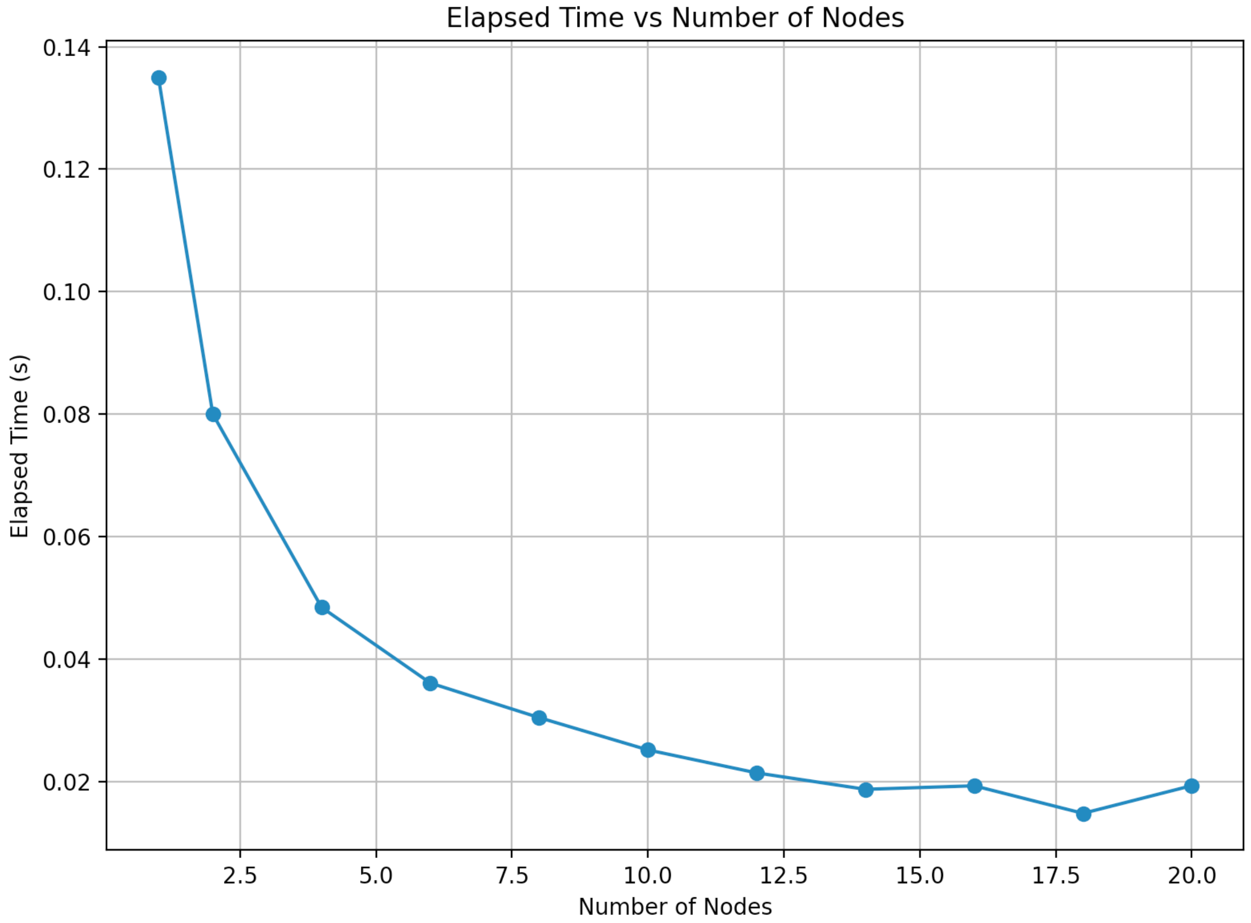
Each node has 1 CPU

N = 4294967 (1 task per node)



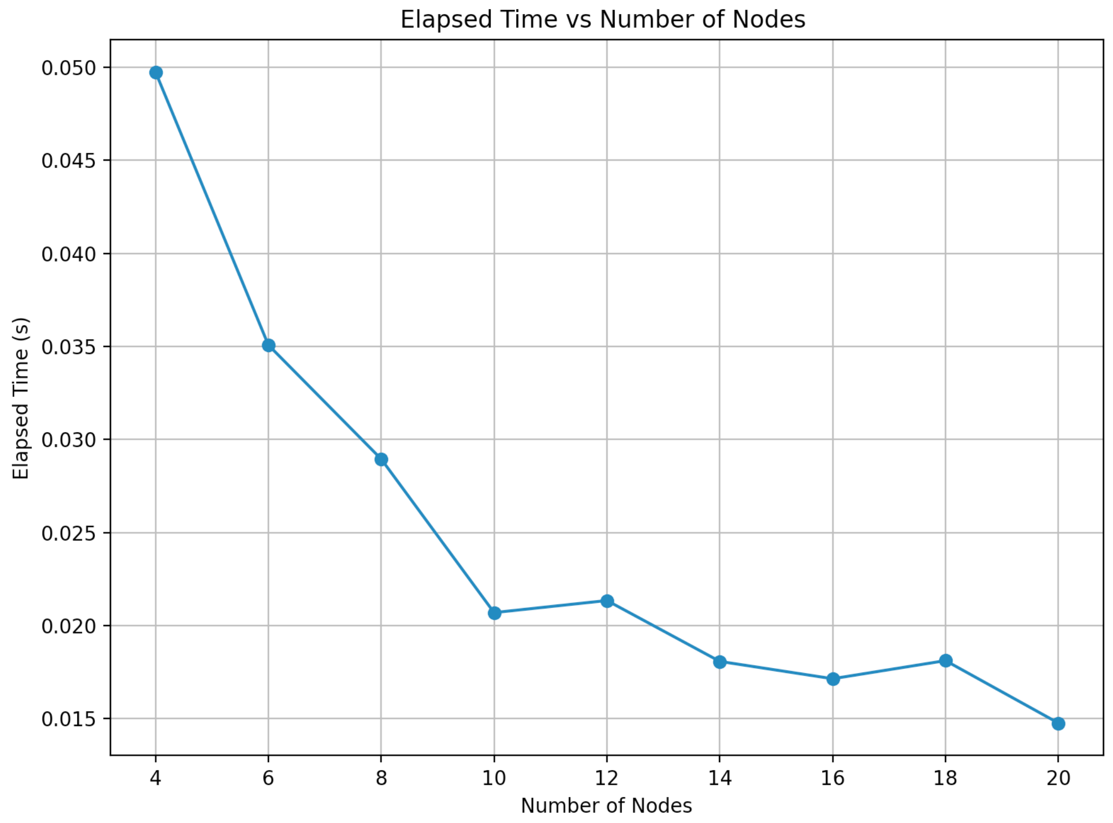
Each node has 1 CPU

N = 42949672 (1 task per node)



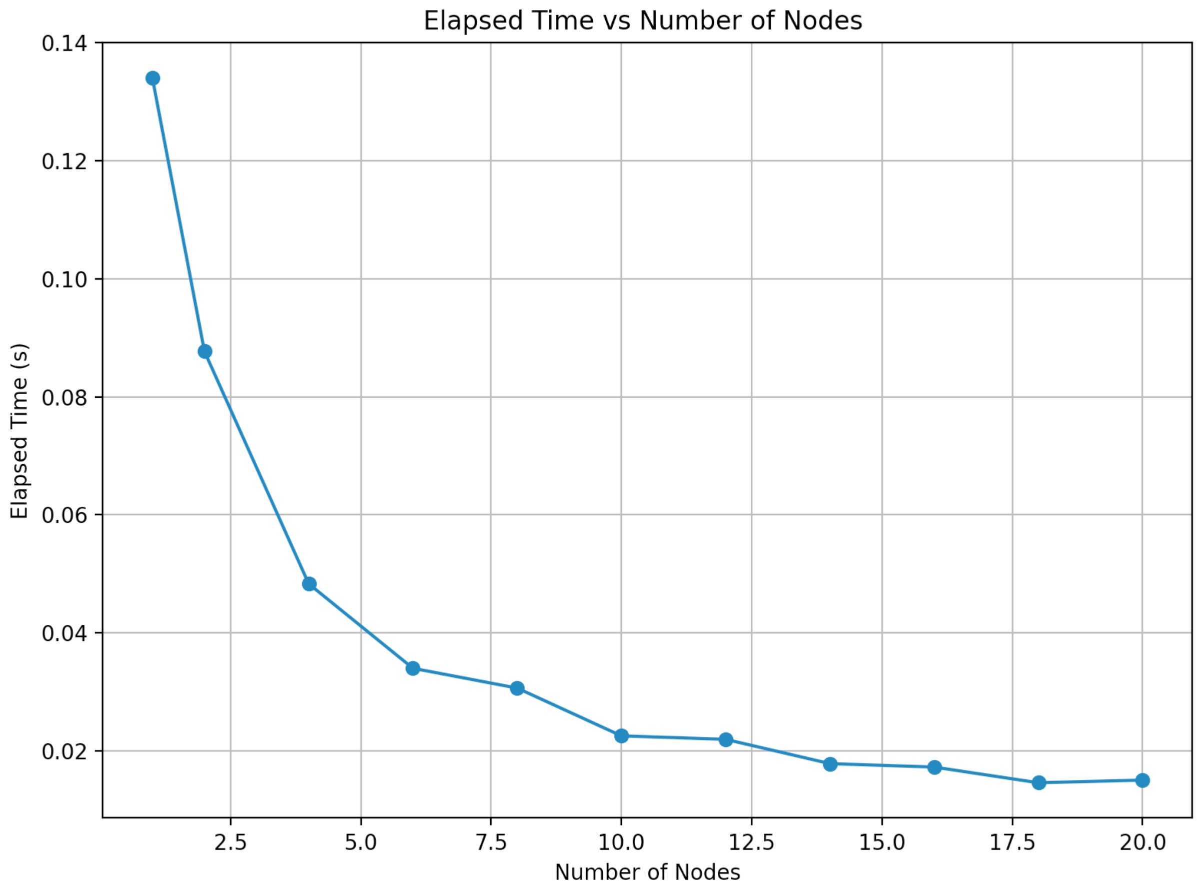
Each node has 1 CPU

N = 42949672 (2 task per node)



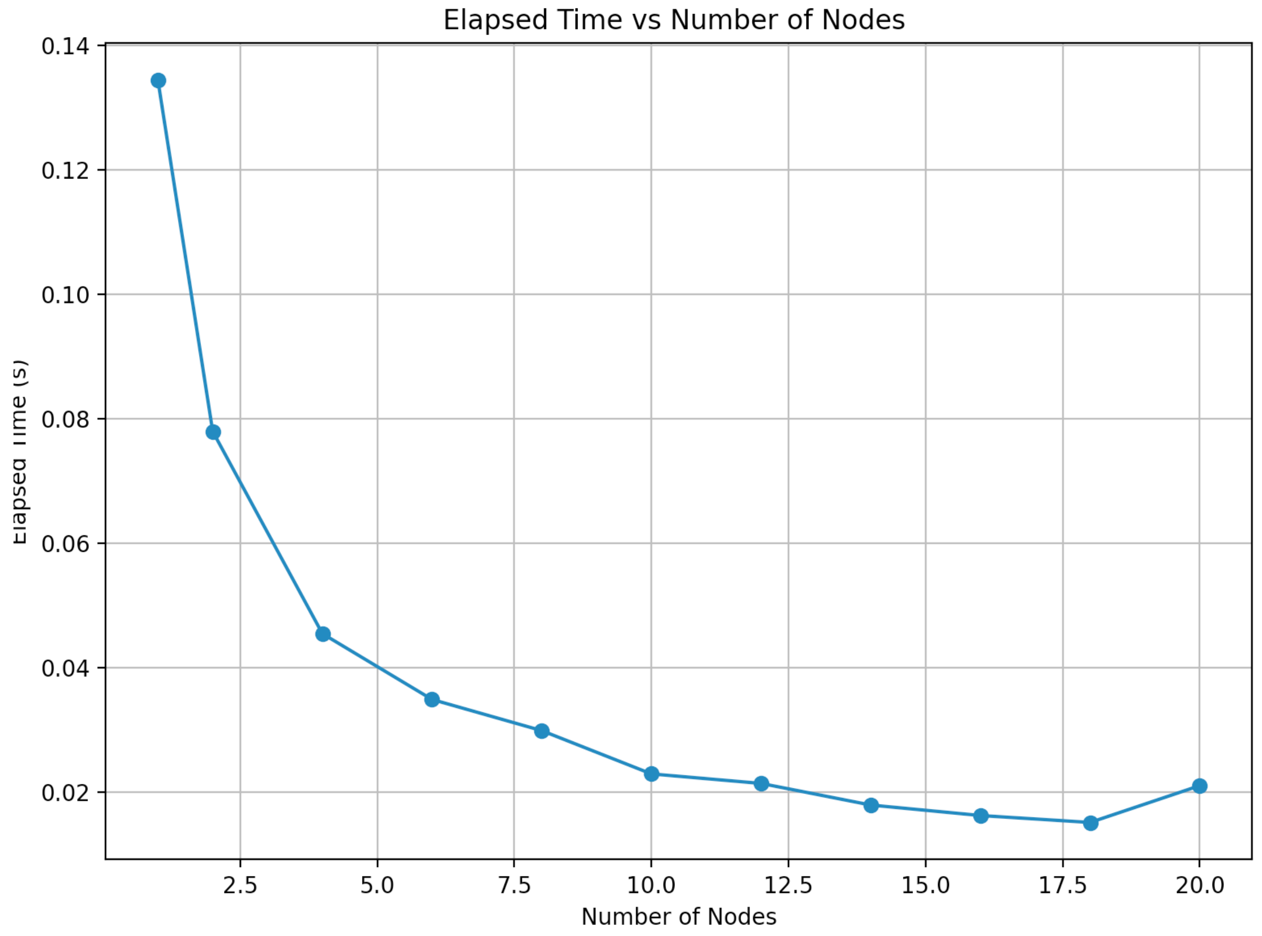
Each node has 2 CPUS

N = 42949672 (4 task per node)



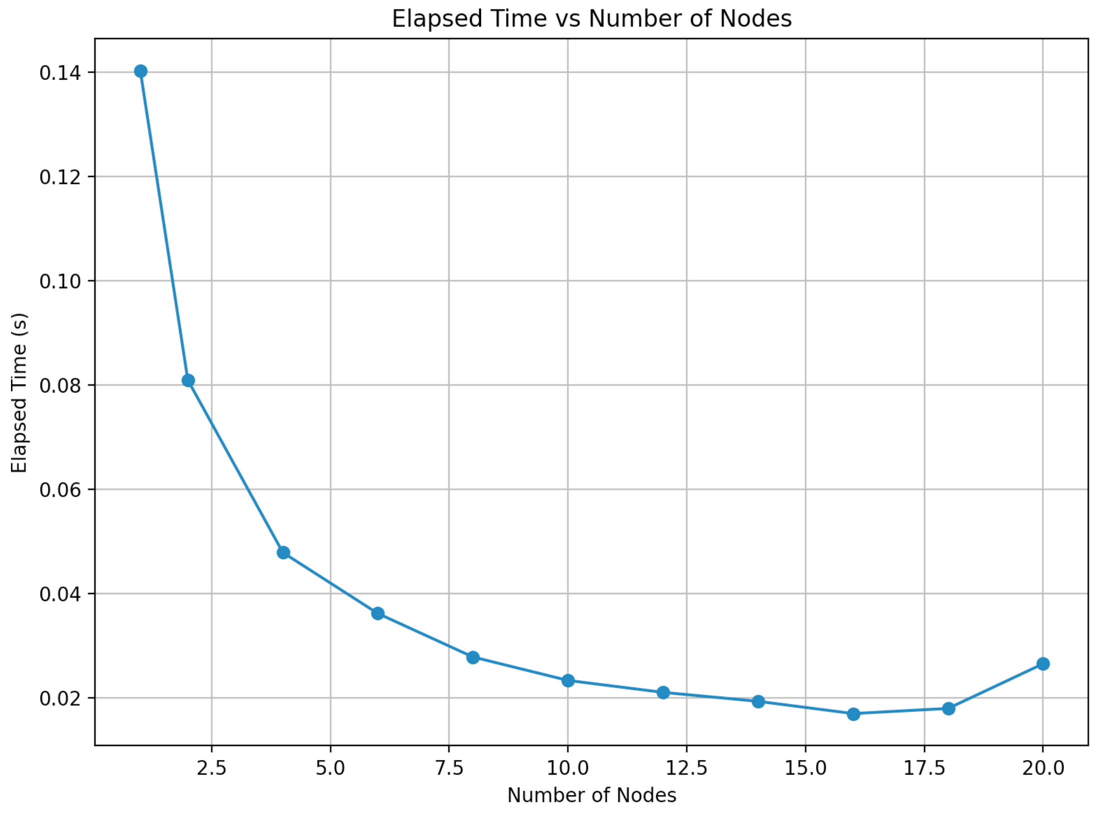
Each node has 4 CPUS

N = 42949672 (6 task per node)



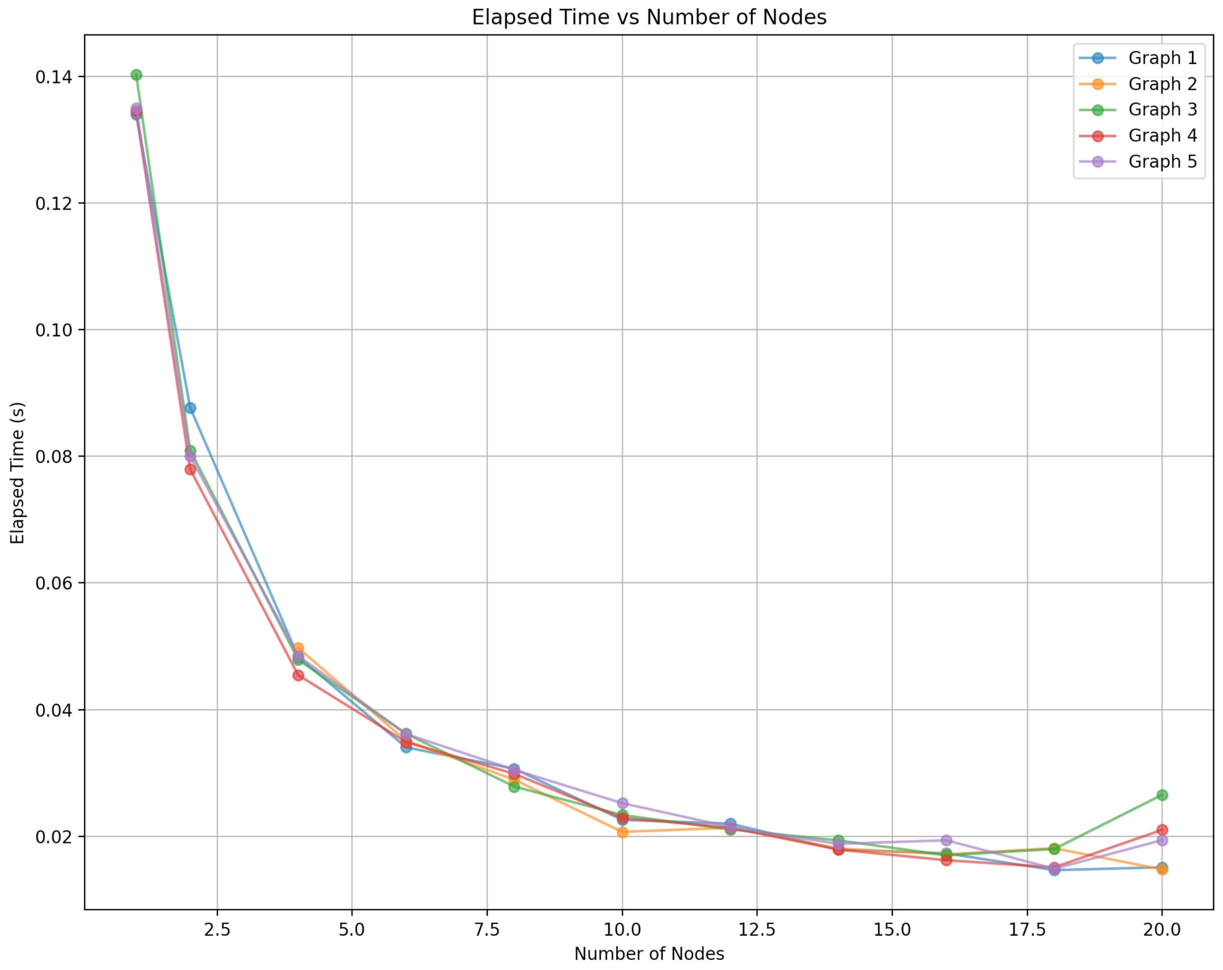
Each node has 6 CPUS

N = 42949672 (8 task per node)



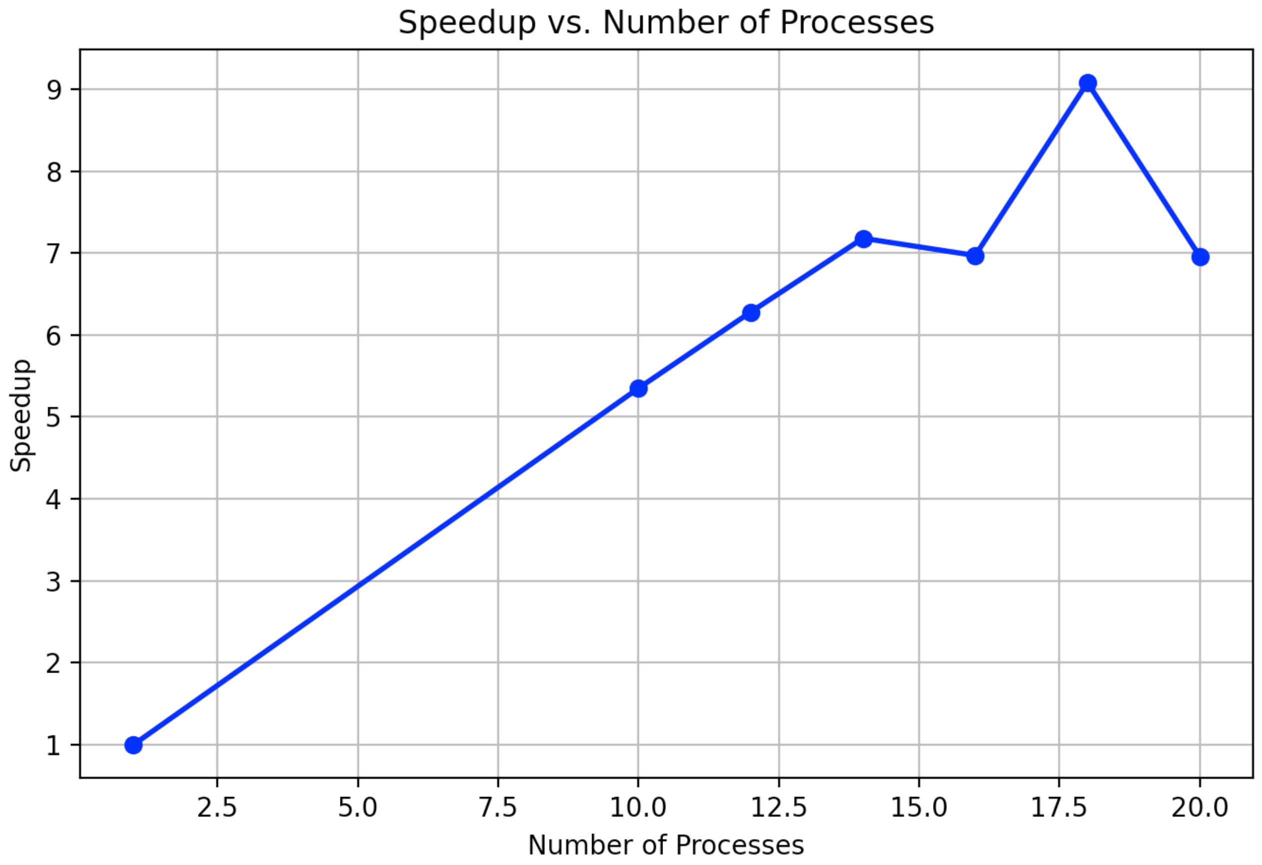
Each node has 8 CPUS

N = 42949672


Elapsed Time vs Number of Nodes
Elapsed Time (s)
0.14
0.12
0.10
0.08
0.06
0.04
0.02
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
Number of Nodes
Graph 1
Graph 2
Graph 3
Graph 4
Graph 5

N = 42949672

- Sequential Execution Time (for 1 process) = 0.135017 seconds



Summary

We were able to see that there is a small bottleneck after a certain threshold in the number of processors. The benefit gained on time saved decreases as we increased number of processors beyond a threshold.

Thank you

Kiran Radhakrishnan

kiranrad@buffalo.edu