# Parallel Implementation Of Merge Sort

Authors :
Shalin : 201301415
Samarth : 201301456

# Problem Introduction

- Merge sort is a Divide and Conquer algorithm. It divides input array into two halves, calls for itself for the two halfs and merge the two sorted halves.
- So the time complexity equation would be like this:

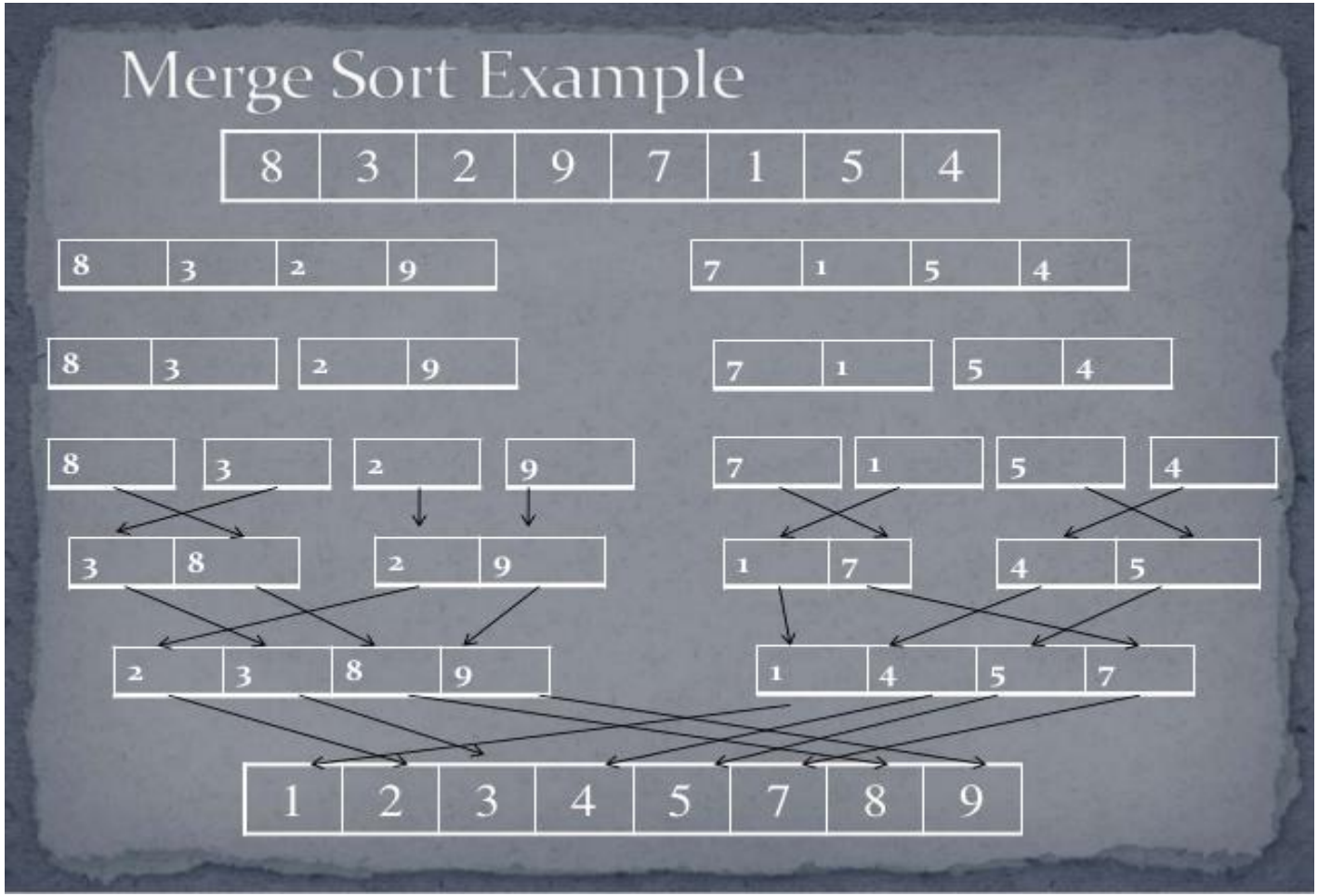
  $T(n) = 2*T(n/2) + O(n)$

- It consumes O(n) memory (space) for serial code.
- References :

  http://geeksquiz.com/merge-sort/

  https://en.wikipedia.org/wiki/Merge_sort

  https://www.youtube.com/watch?v=TzeBrDU-JaY


Merge Sort Example
8 3 2 9 7 1 5 4
8 3 2 9
7 1 5 4
8 3
2 9
7 1
5 4
8
3
2
9
7
1
5
4
3 8
2 9
1 7
4 5
2 3 8 9
1 4 5 7
1 2 3 4 5 7 8 9

# Applications

- Gmail’s displays of email in inbox records the mail by their receive date, which is based on sorting algorithm like merge sort and quicksort.
- Facebook , Quora , Twitter feed (newsfeed page) is also uses the sorting algorithm.
- Sites like amazon ,flipkart which is used heavily by peoples also uses the sorting algorithm to give the user what he/she wants.
- To order the marks of exam, sorting is used.
- In sort, sorting is everywhere around us.

# Serial Algorithm (pseudo-code)

**Recursive Implementation**

**MergeSort(arr[],left,right)**

If r>l

Find the middle point to divide the array into two halves:

middle mid =(left+right)/2

Call mergeSort for first half:

**mergeSort(arr,left,mid)  (Can do parallely)**

Call mergeSort for second half:

**mergeSort(arr,mid+1,right)  (can do parallely)**

Merge the two halves sorted in above two steps:

merge(arr,l,m,r)

**Iterative Implementation (Bottom-up merge sort)**

**MergeSort(arr[])**

m=1,n=length(arr)

while(m<n)
  i=0
  **while(i<n-m) do  (can do parallely)**
    merge(arr[i to i+m-1] and arr[i+m to  min(i+2*m-1,n-1)])
    i=i+2*m
  m=m*2

# Complexity of serial code

- From the equation : $T(n) = 2*T(n/2) + O(n)$

  Where,

  $T(n/2)$ is the time for dividing the half subarray and merging that halves.

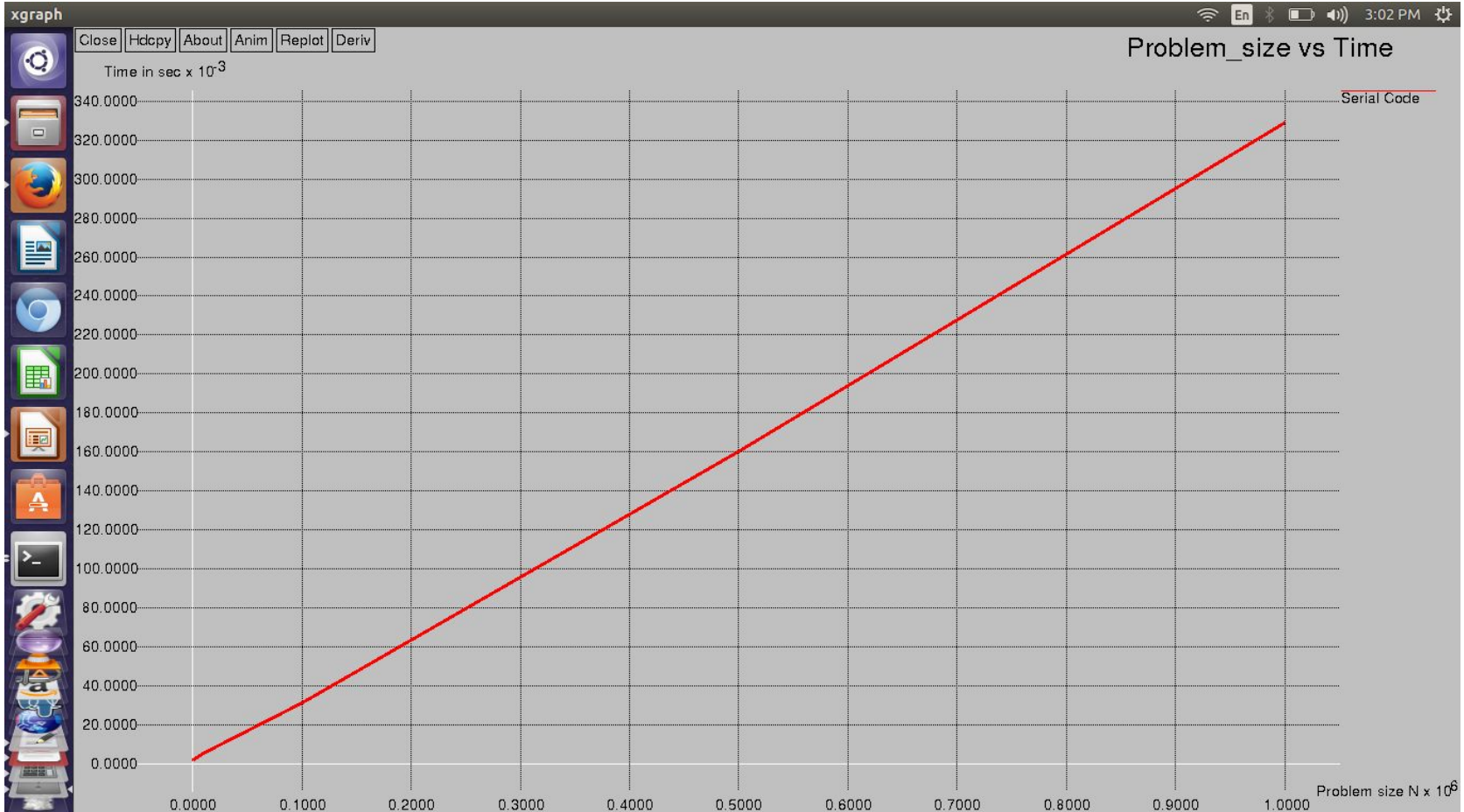
  And $O(n)$ is the time for merging the two half subarray into one.

- We can derive that the time complexity of merge sort is $O(nlogn)$.

# Scope of Parallelism

- As you can see in recursive implementation of merge sort each half can parallely run the **mergeSort()** method.
- In the serial implementation of merge sort you can merge all the subarray of given length parallely, because merging them can run independently.
- In that case, we can parallelize the inner while loop (check the above slide).
- Parallelizing the serial code is much easier and time consuming than parallelizing the recursive code.
- Here In both the cases, scanning and writing the array from/in the file is done serially.

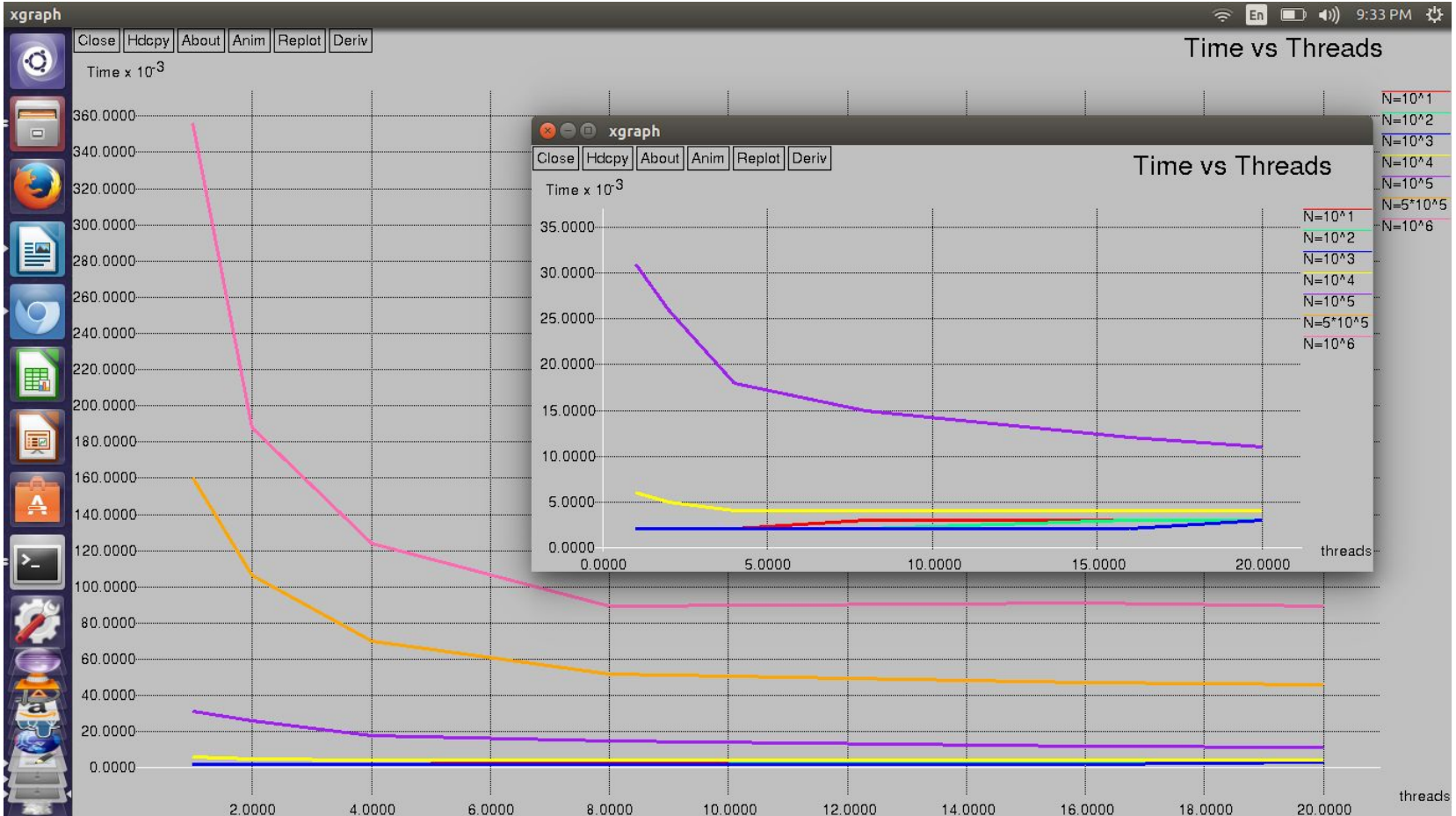
# Problem size (N) vs Time



# Explanation

- Complexity of merge sort is O(nlogn). So as we increase the problem size, time taken (to sort the given array) increases.
- In the above graph problem size (N) is increasing from 10^1 to 10^6. And time to sort the array is also increases as we increases the problem sizeN.

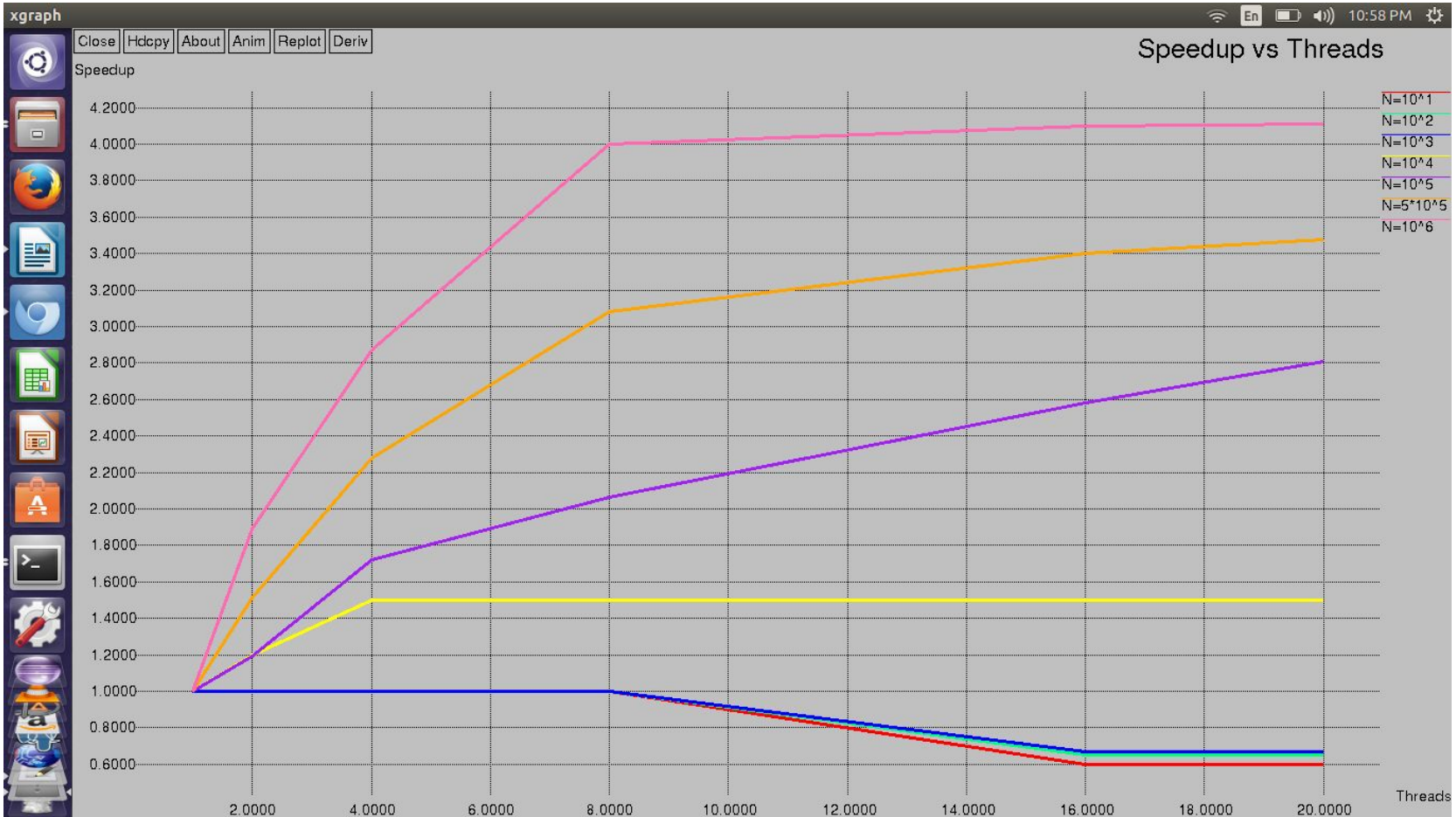
# Threads vs Time (for diff. value of N)



# Explanation

- For a large problem size(N) , as the number of threads increase time taken by the parallel code decreases.
- After a point, the time doesn’t change much as we increases the threads.
- This is because of overhead.
- For  a small problem size(N), it is better if we use serial code. Because the time doesn’t vary much as we increases the threads.
- And you can see from the above graph, that after a time increases as we increase the no. of threads.

# Speed up curve (As a function of no. of processors)



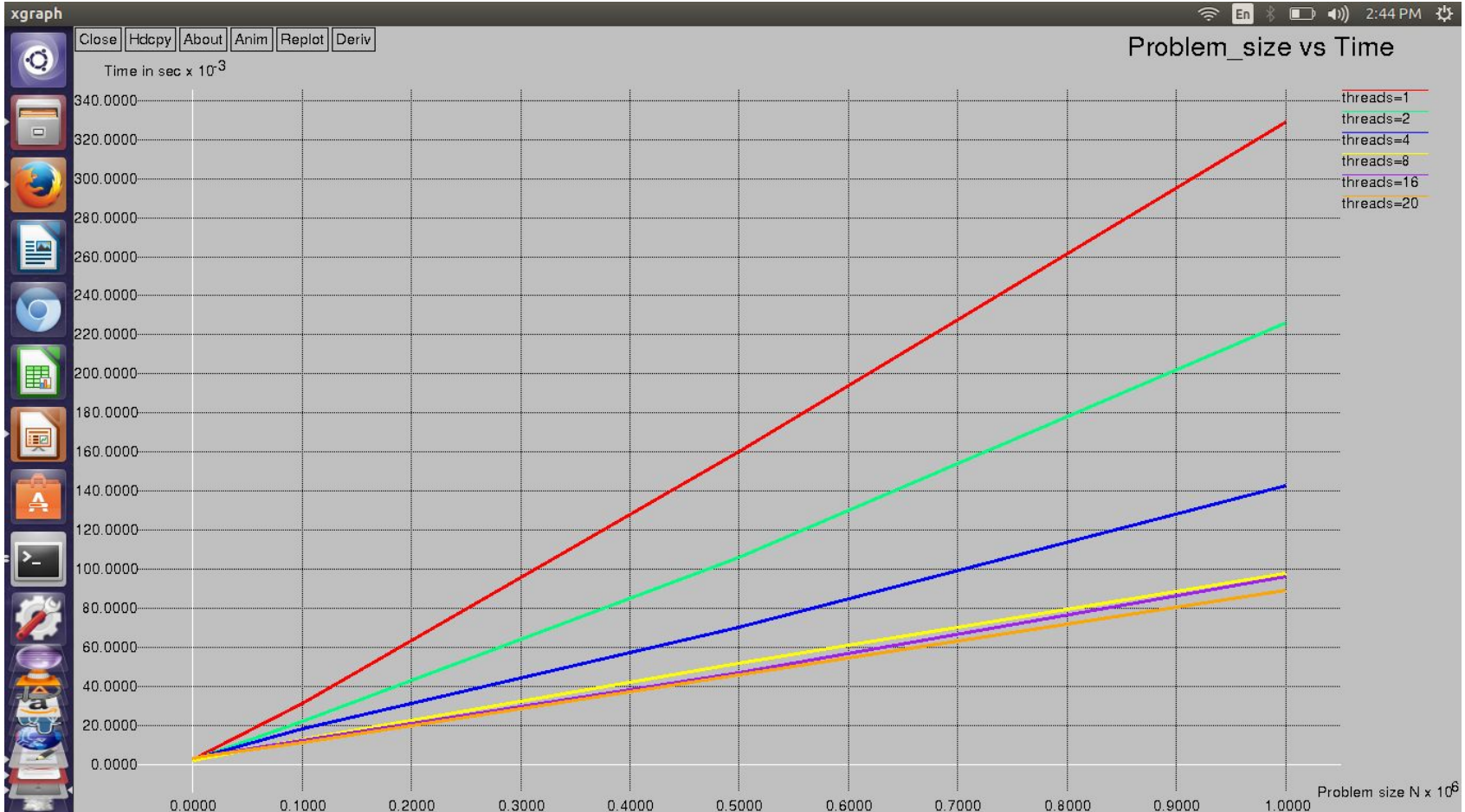
# Explanation

- For N = 10^6

| Processors | 2 | 4 | 8 | 16 | 20 |
|---|---|---|---|---|---|
| Speedup | 1.89 | 2.87 | 4.00 | 4.10 | 4.11 |
| e | 0.0582 | 0.131 | 0.143 | 0.193 | 0.258 |

- Since e is steadily increasing as the number of processors increases ,the main reason for the poor speedup is overhead.

# Problem size(N) vs Time (For diff. no of threads)



# Speedup vs Problem size(N) (for diff. no of threads)

# Conclusion And Scope of Improvement

- As we increase the problem size, the speedup increase for given no. of processors.
- For No. of threads = 8:

| Prob. Size | 10 | 100 | 1000 | 10000 | 100000 | 500000 | 1000000 |
|---|---|---|---|---|---|---|---|
| speedup | 1 | 1 | 1 | 1.5 | 2.067 | 3.08 | 4.00 |

- For a small problem size serial code is better than parallel.
- Only Scanning/writing array from/in file is done serially. So if we can scan/write from/in file parallely, we can get much higher speedup.
- But still overhead would always be there.