COL331

LAB ASSIGNMENT – 2

Submitted By :- Nikita Bhamu
Entry number :- 2018CS50413

Approach implemented to find whether the number is prime or not :-

- For this I have used the basic approach which traveses number k from 2 to sqrt(N) and finds if (N mod k) == 0 or not, if it is then the number is not prime else it is.

Approches implemented to parallelise the program using multi-threading :-

- First of all, to implement the multi-threading the pthreads library of C is used.

1. **Naïve approach ->**
   In this approach I divided the domain [2…N] into t partitiones in which first t-1 partitions have size (N-1/t) and the last one has size (N-1/t + extra numbers) , where t is the number of thread. The first (t-1) threads gets exactly (N-1/t) number to look at whether they are prime or not.

- **I used the pthread_create to create the threads.**
  For implementing this, I created a structure named naïve_thread_data which contains the following attributes :-
  1. **Domain** :- Domain is an array pointer of which points an array of size two which contains the starting and ending number of the partition given to a particular

thread.
2. **Arr** :- Arr is basically an array pointer which pints to an array of size N having all the entries 0, which represent that all the number are initiliased as NOT PRIME initially and as the program executes and if the threads find the number to be prime then the corresponding entry of this array is made 1.
3. **Time** :- Time is an array pointer which pints to an array of size num_threads which will finally store the time for which a thread is executed.
4. **Threadid** :- It iis an integer which contains the number given to that particular thread.

2. Load balanced ->
In this approach the threads take a number dynamically from the number array and check it whether it is prime or not and update the corresponding result in the arr which cntains the booleans(0,1) which denote whether the number is prime or not.

- **I used the pthread_create to create the threads.**
- **To ensure that if a number is checked by 1 thread or a thread is checking that number at present then any othe thread doesn't alter it, we use the locks provided by the pthread header.**
  Lock creation :
  pthread_mutex_t mutex =
  PTHREAD_MUTEX_INITIALIZER
  Locking it:
  pthread_mutex_lock(&mutex);

Unlocking it:

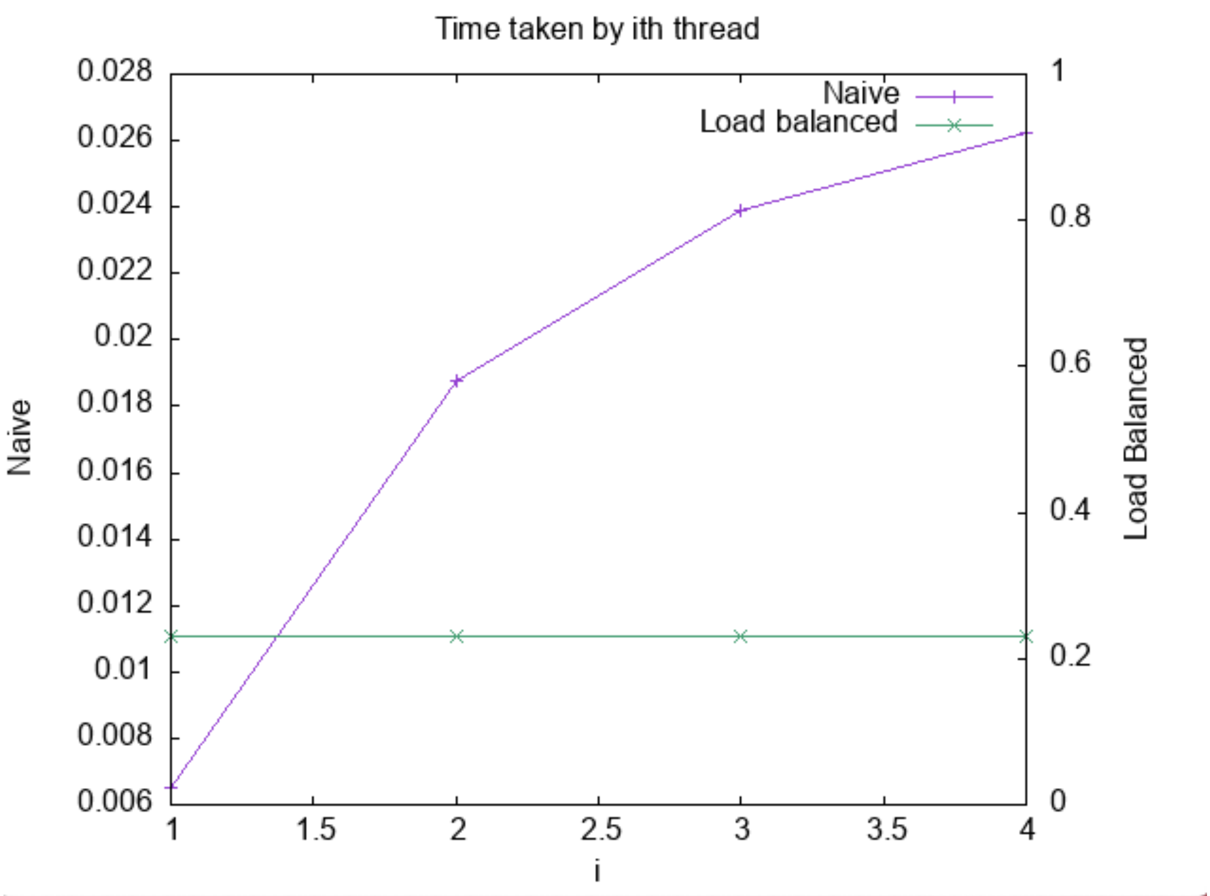
```
  pthread_mutex_unlock(&mutex);
```

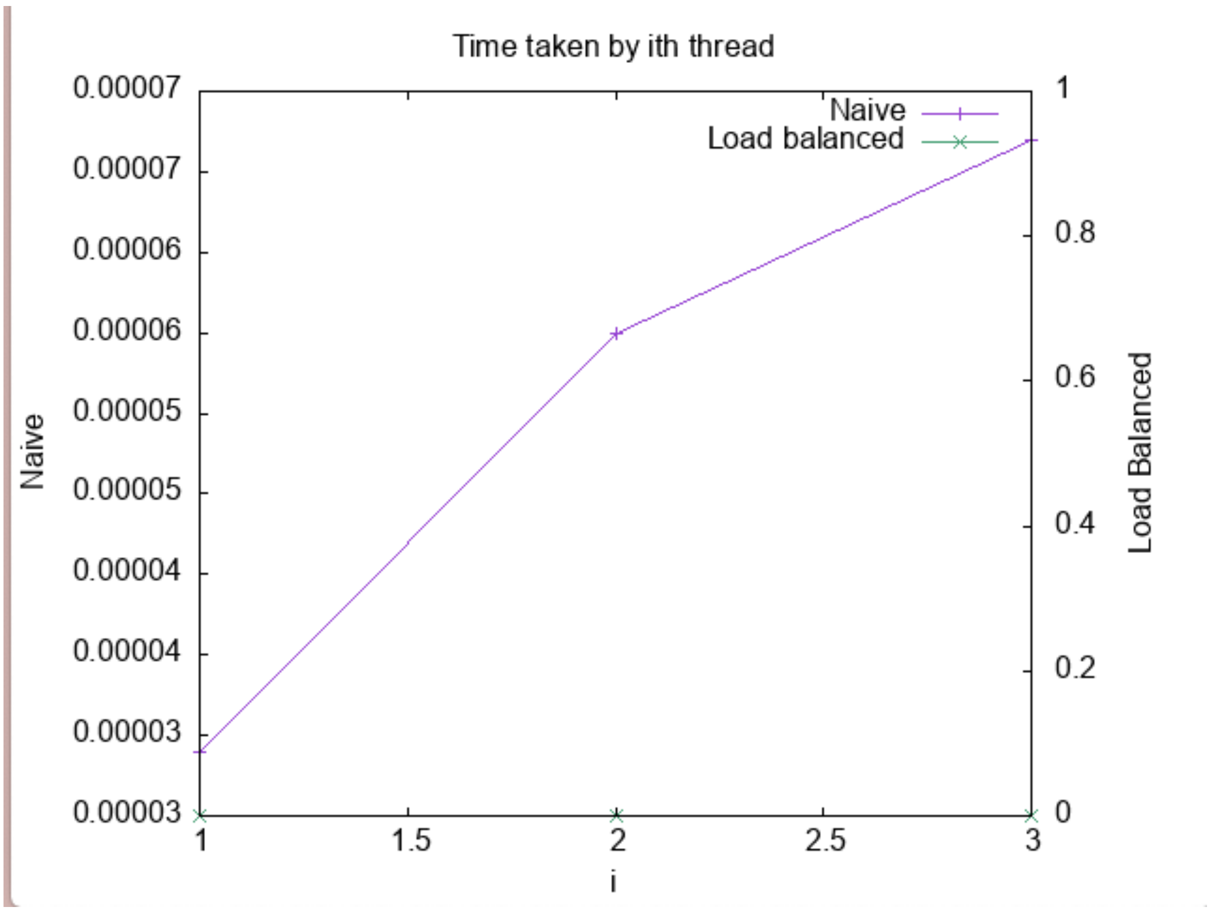
For implementing this, I created a structure named loadbalanced_thread_data which contains the following attributes :-

1. **Counter** :- This is a pointer which points to the integer containing the count upto which the number have been looked at by the thread, now when a new thread will look at the array it will check for the number to which counter is pointing at present which will be the velue of (*counter+1). In this way none of the number will be checked twice by the threads and everythread will have a counter to tell them there next prey.
2. **Arr** :- Arr is basically an array pointer which pints to an array of size N having all the entries 0, which represent that all the number are initiliased as NOT PRIME initially and as the program executes and if the threads find the number to be prime then the corresponding entry of this array is made 1.
3. **Time** :- Time is an array pointer which pints to an array of size num_threads which will finally store the time for which a thread is executed.
4. **Threadid** :- It iis an integer which contains the number given to that particular thread.
5. N :- N basically contains the number upto which we have to check prmes. It is used in the way that when threads are checking the number which the counter’s value is telling them but if the number exceeds N , then threads must exit, hence we took it as an attribute.

CONCLUSIONS FROM THE ASSIGNMENT :-

> Some of the plots which I got :





- **We can conclude from the graphs that the time taken by threads in the load balanced approach is almost equal. This is because the load balanced approach allocate the number to the thread dynamically. Hence, on a bigger picture we can see all the threads get number of almost same values , i.e., initially all threads will have small numbers to check whether they are prime or not and during the end all will have larger number to check and this continues throughout the process.**

- **The time taken by the threads in naïve approach fluctuates and often it increases , because the threads coming first gets all the numbers smaller than the thread coming after it, hence often time increases as**

the thread_id increases but it may show some amount of fluctutation as well.