Name : Rakesh Baingolkar
UB : 50097576

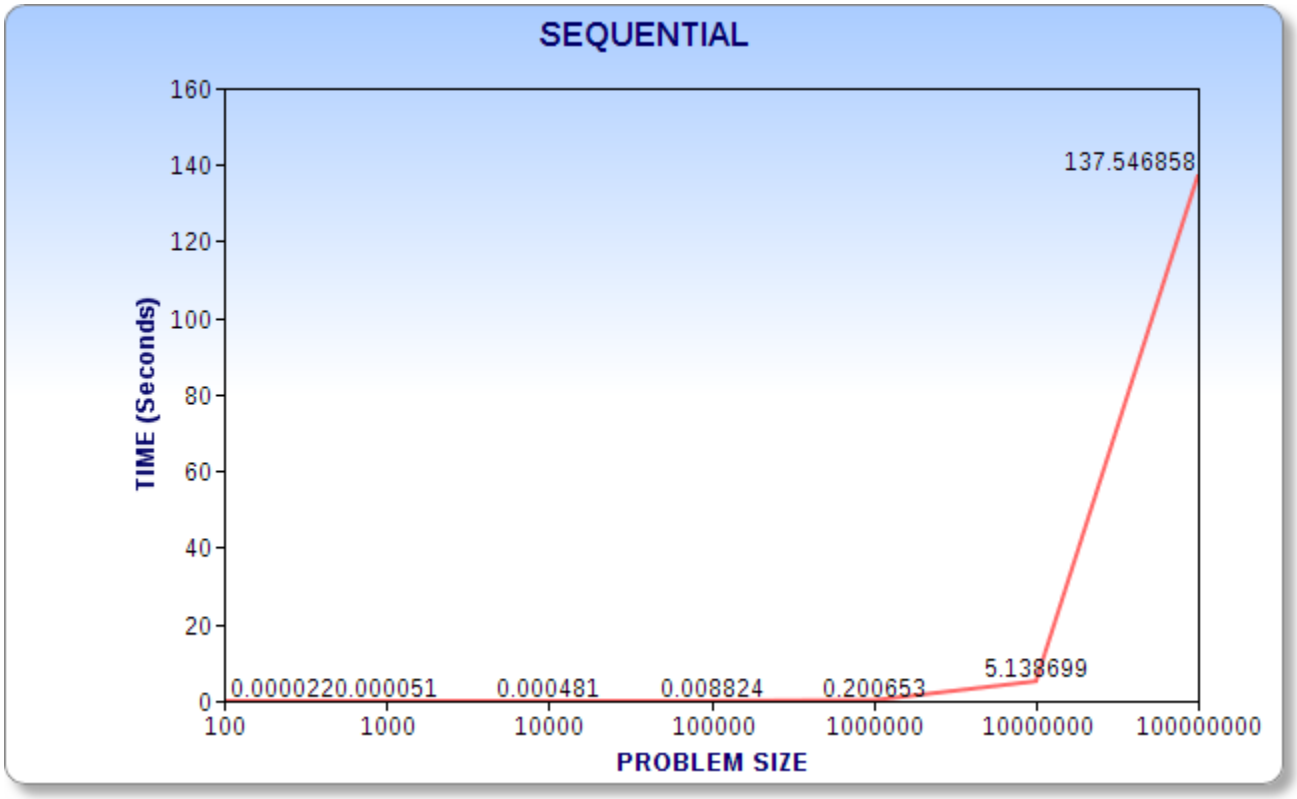
PROGRAMMING ASSIGNMENT : 3

PRIME/

SEQUENTIAL:

File:Sequential.c
makefile: Makefile
Run makefile: make
Object file: Sequential
Run the Object file: ./Sequential -t Problem_size

The following graph shows the time values with increase in problem size.



We see that the time taken increases with problem size.

GPU:

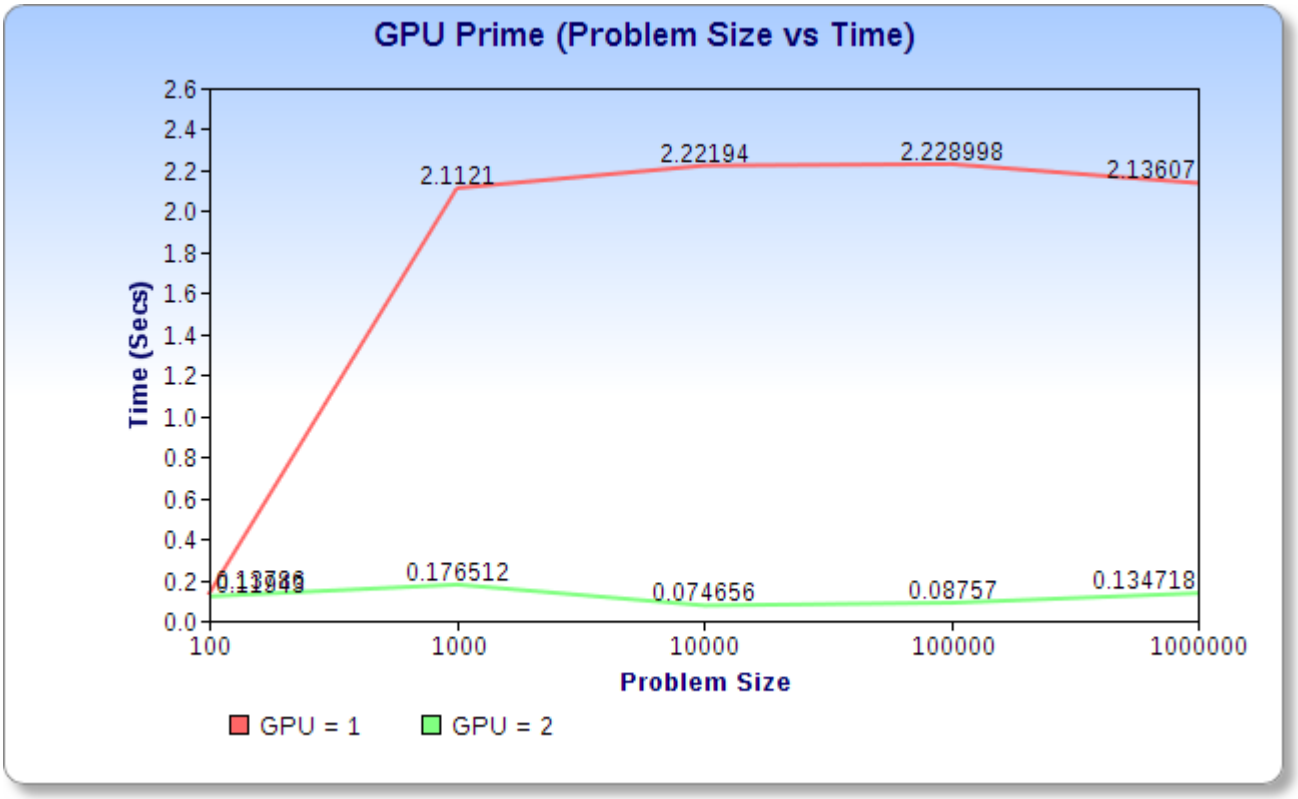
File : GPU_Prime.cu
makefile: Makefile
Run makefile: make
Object file: GPU_Prime
Run the Object file: ./GPU_Prime -t Problem_size

Rationale:

I stored the prime numbers in an array (A) and the square roots in array (B) and passed these arrays to the Kernel function along with an array (c) indicating if the number is prime of not. The square root array was passed to indicate the limit upto which we will check the divisibility of a number.

For the CUDA function I used N blocks with one thread I.e each block will take care of a number and thus all the blocks will run in parallel making the process efficient.

I increased the problem size for a function for GPU = 1 and then for GPU = 2. The results were as follows.



From the graph we can see that as we increase the GPU then the time significantly decreases. Also one more observation can be that in case of 1 GPU when we increase the Problem Size the time taken increases and in the case when we use 2 GPU then the time taken more or less remains constant for increasing Problem Size.

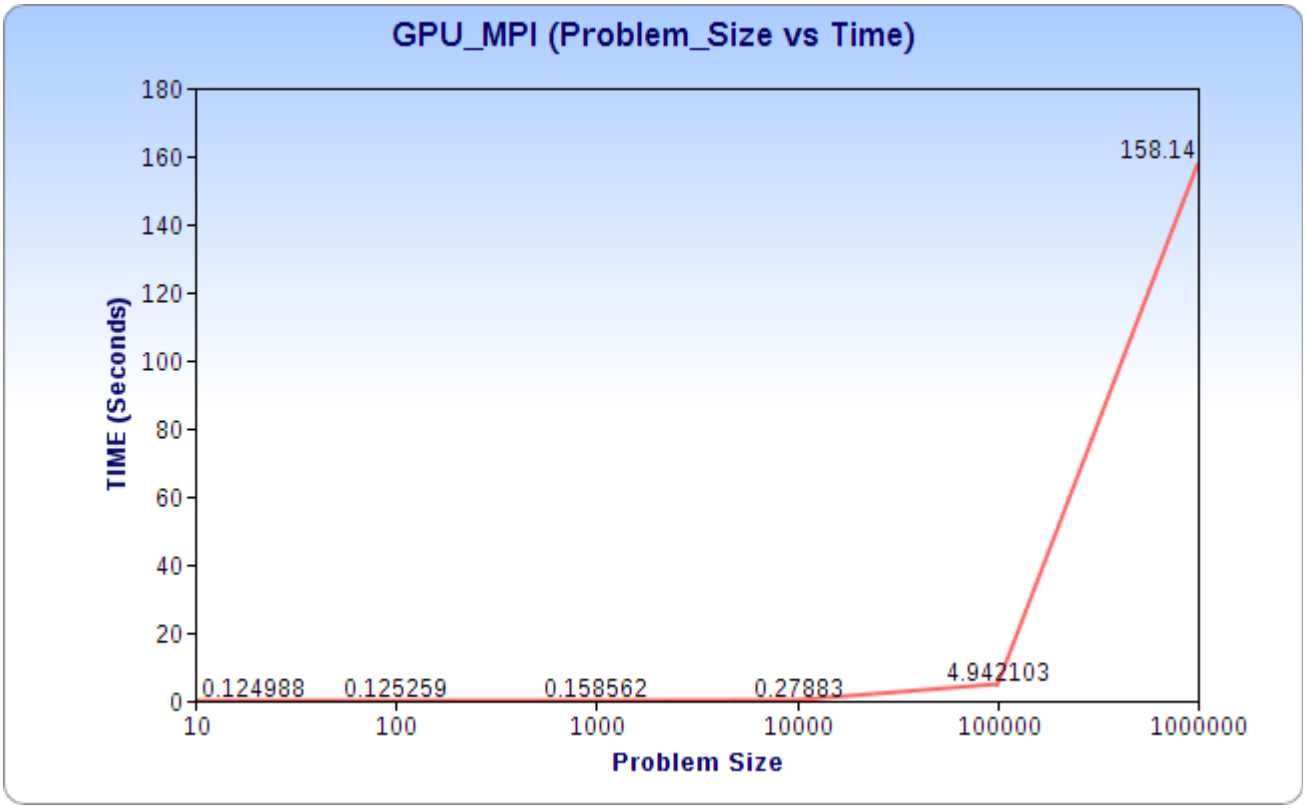
GPU_MPI:

MPI File : MPI+GPU.c
CUDA File : CUDA.cu
makefile: Makefile
Run makefile: make
Object file: MPI_CUDA_PRIME
Run the Object file: ./MPI_CUDA_PRIME -t Problem_size

Rationale:

In GPU MPI I divide the total problem size equally to number of cores and then each core invokes the cuda function with the range given to it by the master. The function returns the count and the largest values in a 2 element array. These values are compared by the master to find the total count and the largest primt number.

I tested the code using mpirun command which is technically not running on multiple nodes but as CCR was very busy I went ahead with it. The observation is as follows:



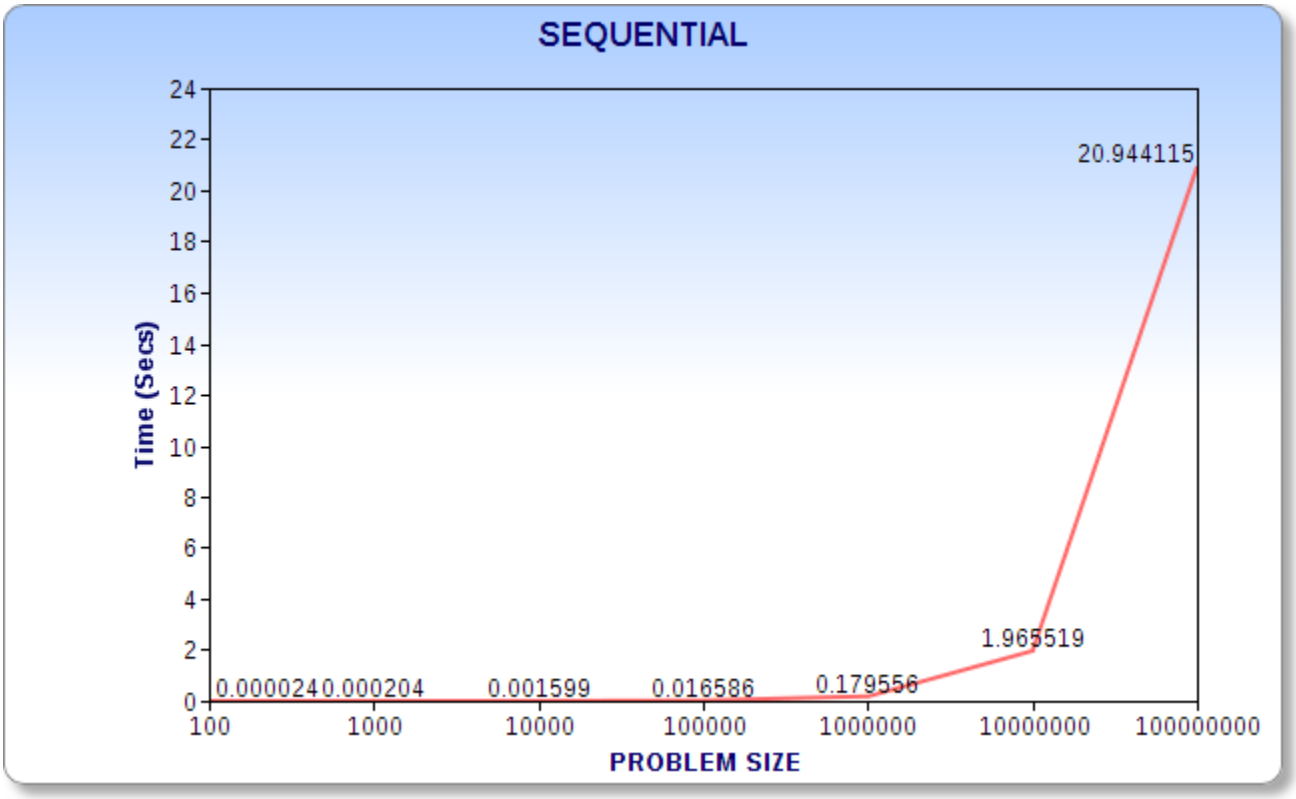
We see that the time increases with increase in problem size. Nodes were 2 for all the readings.

SORT/

SEQUENTIAL:

File : Sequential.c
makefile: Makefile
Run makefile: make
Object file: Sequential
Run the Object file: ./Sequential -t Problem_size

The following graph plots the time taken vs the problem size.



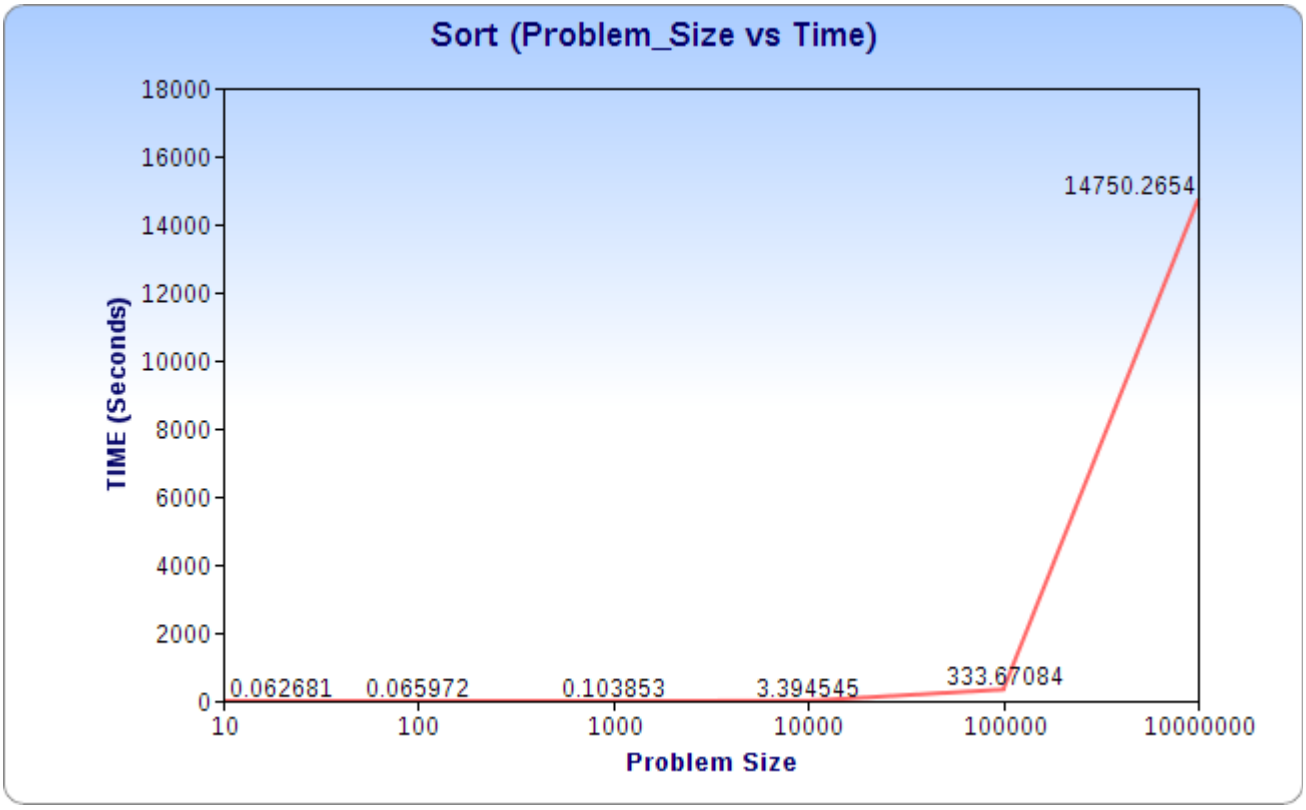
From the graph we can see that the time taken increases with the problem size.

GPU:
File : GPU_Sort.cu
makefile: Makefile
Run makefile: make
Object file: GPU_Sort
Run the Object file: ./GPU_Sort -t Problem_size

Rationale:

I divided the input array into number of buckets and then I sent the all the buckets to the kernel function with the block number as the number of buckets. The kernel function will run once for each bucket I.e it will sort all the elements in a single bucket and return the bucket. The parallel sorting of buckets is what makes this function faster.

I increased the problem size for a function for GPU = 1.


Sort (Problem_Size vs Time)
TIME (Seconds)
18000
16000
14000
12000
10000
8000
6000
4000
2000
0
0.062681
0.065972
0.103853
3.394545
333.67084
14750.2654
10
100
1000
10000
100000
10000000
Problem Size

Program	Number of cores and device	Problem Size	PRIME	SORT
Sequential	1 Node 1 task-per-node 0 GPU	100000	0.008824	0.016586
Sequential	1 Node 1 task-per-node 0 GPU	1000000	0.200653	0.179556
GPU	1 Node 1 task-per-node 1 GPU	100000	2.228998	333.670844
GPU	1 Node 1 task-per-node 1 GPU	1000000	2.13607	14750.2654
GPU_MPI	1 Node 1 task-per-node 2 GPU	100000		
GPU_MPI	2 Node 2 task-per-node 2 GPU	100000		
GPU_MPI	2 Node 4 task-per-node 2 GPU	1000000		
MPI	2 Node 4 task-per-node 0 GPU	1000000	54.940667	3.799325
OPENMP	1 Node 8 task-per-node 0 GPU	1000000	21.213528	2.439565