Team: Tejeshwar Induri, Bhhaanu Pirange, Harika Battu

PROBLEM STATEMENT: Finding the largest prime using Quick Sort with parallel programming.

Description:

This project is about implementing a sequential and two parallel programs to find the largest prime number in each set of data using sorting techniques. Although we have the sorting technique implemented programs in different languages, our goal is to combine those with prime number check and use them for parallel programming such that these larger tasks can be carried out simultaneously.

Initially a list of elements is provided which are unsorted. Using our algorithm, the list of sorted elements is processed to find the largest prime from the list.

For sorting: Here, we are going to use quick sort which works on divide and conquer technique by selecting a 'pivot' element from the array and partitioning the other elements into two sub-arrays, according to whether they are less than or greater than the pivot.

Using sorting that is explained above to find the largest prime in the data will be the final task of this project using prime number check logic.

Tasks:

1. Writing a working sequential program for the problem statement and note the execution time for it.
2. Using the sequential program, implementing the parallel programs using parallel programming frameworks i.e., MPI and openmp.
3. Compare and analyze the performances of all the three programs.

Language being used: C programming for all three programs.

Aim: Goal of this project is to show how we can turn down the execution times of sorting to find largest prime using two parallel programs using parallel programming frameworks.

Implementation:

A. We developed a sequential program that performs sorting on given numbers and find the largest prime from it and noted the execution time.
B. Developed an MPI version that specifies a parallel computing model in which each parallel process has its own local memory and data by sending messages between processes.
C. Finally, wrote the openmp program which introduced parallelism into our sequential program by launching a set of threads that execute portions of the code concurrently using compiler directives.

Steps:

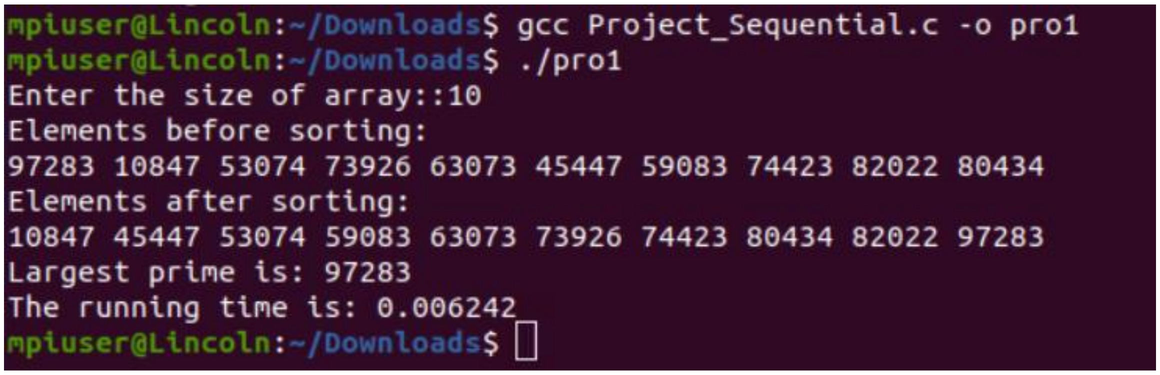
- Using generateRandom() we generated random numbers ranging from 0 to 99999. (Which can be extended as per user requirements)
- To print the generated elements and the sorted list we used ele_list()
- For sorting the random numbers generated we are using quick sort technique, sending the generate random numbers to quicksort() functions sorts them upon recursive calls via partitioning() and swap() functions.
- To find the largest prime from the sorted list we go through every element from the end of the sorted array and check whether it is prime or not. To make this happen we used largestPrime().
- Our code is flexible as it allows users to select the size of the array as per their requirements.

Results:

Sequential Program:

Project_Sequential.c results are as below,

Size can be chosen as per the user requirements

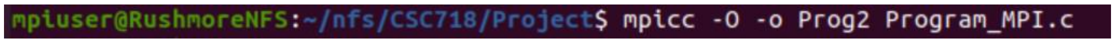


```
mpiuser@Lincoln:~/Downloads$ gcc Project_Sequential.c -o pro1
mpiuser@Lincoln:~/Downloads$ ./pro1
Enter the size of array::10
Elements before sorting:
97283 10847 53074 73926 63073 45447 59083 74423 82022 80434
Elements after sorting:
10847 45447 53074 59083 63073 73926 74423 80434 82022 97283
Largest prime is: 97283
The running time is: 0.006242
mpiuser@Lincoln:~/Downloads$
```

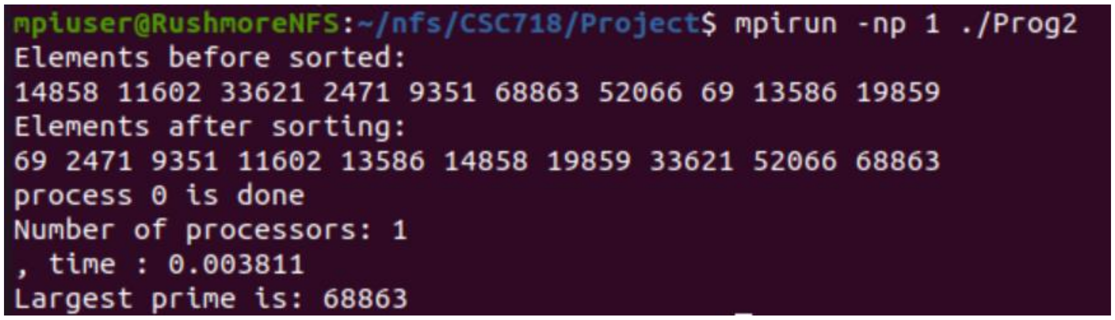
MPI program:

Program_MPI.c results are shown below by considering 10 as the size of array for testing purpose…

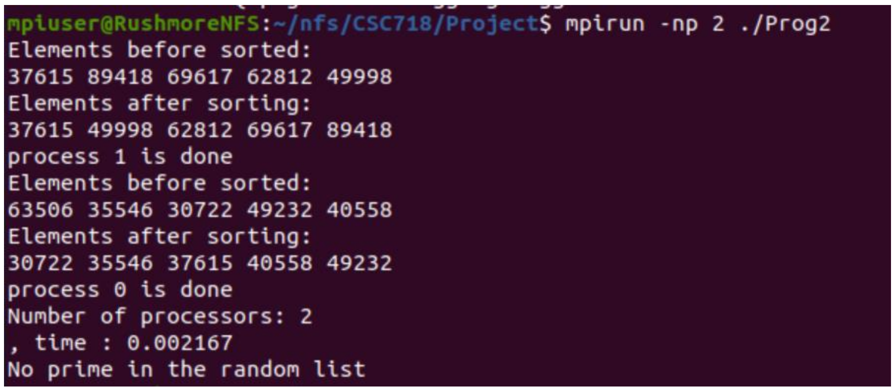
SIZE can be updated according to the user requirements.



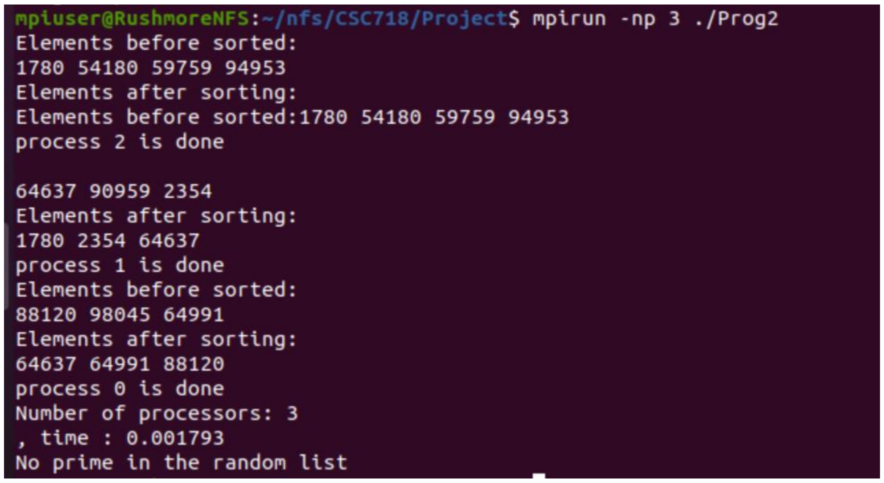
```
mpiuser@RushmoreNFS:~/nfs/CSC718/Project$ mpicc -O -o Prog2 Program_MPI.c
```



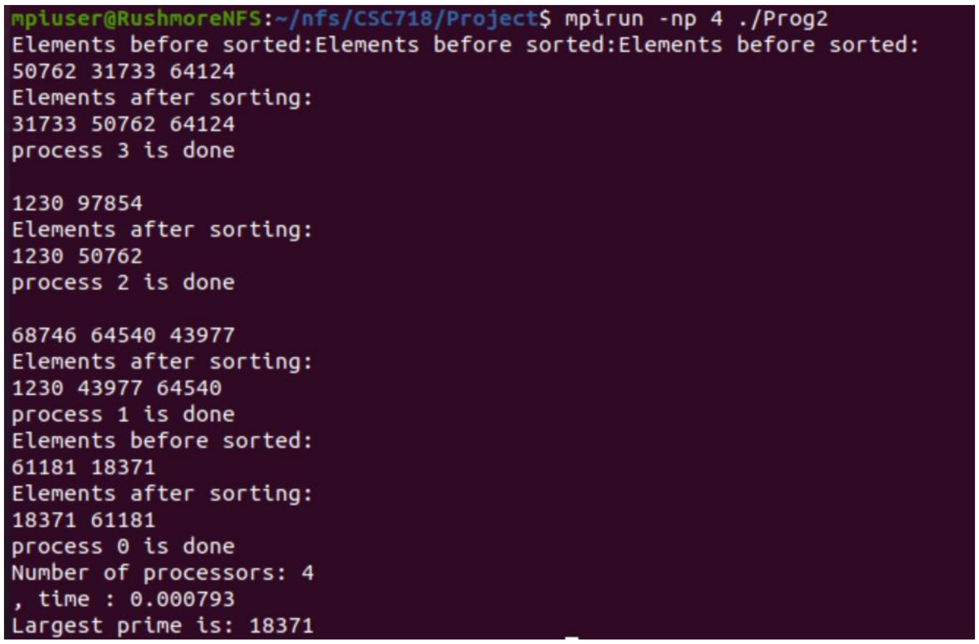
```
mpiuser@RushmoreNFS:~/nfs/CSC718/Project$ mpirun -np 1 ./Prog2
Elements before sorted:
14858 11602 33621 2471 9351 68863 52066 69 13586 19859
Elements after sorting:
69 2471 9351 11602 13586 14858 19859 33621 52066 68863
process 0 is done
Number of processors: 1
, time : 0.003811
Largest prime is: 68863
```



```
mpiuser@RushmoreNFS:~/nfs/CSC718/Project$ mpirun -np 2 ./Prog2
Elements before sorted:
37615 89418 69617 62812 49998
Elements after sorting:
37615 49998 62812 69617 89418
process 1 is done
Elements before sorted:
63506 35546 30722 49232 40558
Elements after sorting:
30722 35546 37615 40558 49232
process 0 is done
Number of processors: 2
, time : 0.002167
No prime in the random list
```



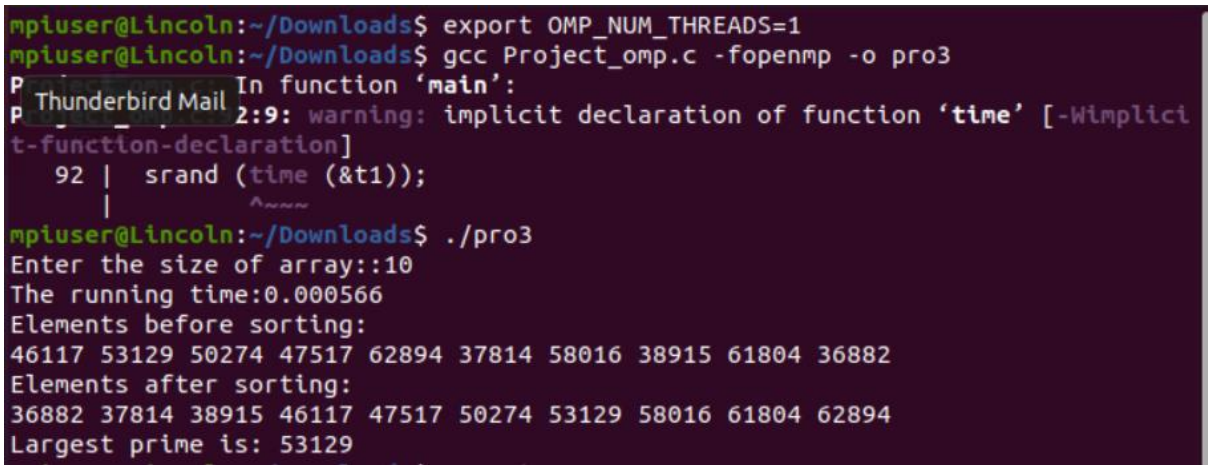
```
mpiuser@RushmoreNFS:~/nfs/CSC718/Project$ mpirun -np 3 ./Prog2
Elements before sorted:
1780 54180 59759 94953
Elements after sorting:
Elements before sorted:1780 54180 59759 94953
process 2 is done

64637 90959 2354
Elements after sorting:
1780 2354 64637
process 1 is done
Elements before sorted:
88120 98045 64991
Elements after sorting:
64637 64991 88120
process 0 is done
Number of processors: 3
, time : 0.001793
No prime in the random list
```



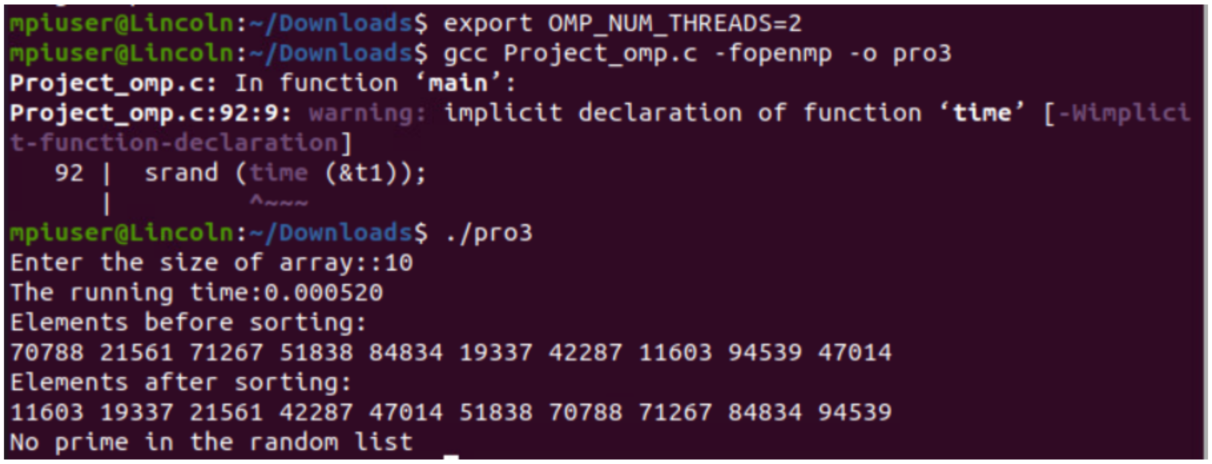
```
mpiuser@RushmoreNFS:~/nfs/CSC718/Project$ mpirun -np 4 ./Prog2
Elements before sorted:Elements before sorted:Elements before sorted:
50762 31733 64124
Elements after sorting:
31733 50762 64124
process 3 is done

1230 97854
Elements after sorting:
1230 50762
process 2 is done

68746 64540 43977
Elements after sorting:
1230 43977 64540
process 1 is done
Elements before sorted:
61181 18371
Elements after sorting:
18371 61181
process 0 is done
Number of processors: 4
, time : 0.000793
Largest prime is: 18371
```

openmp Program:

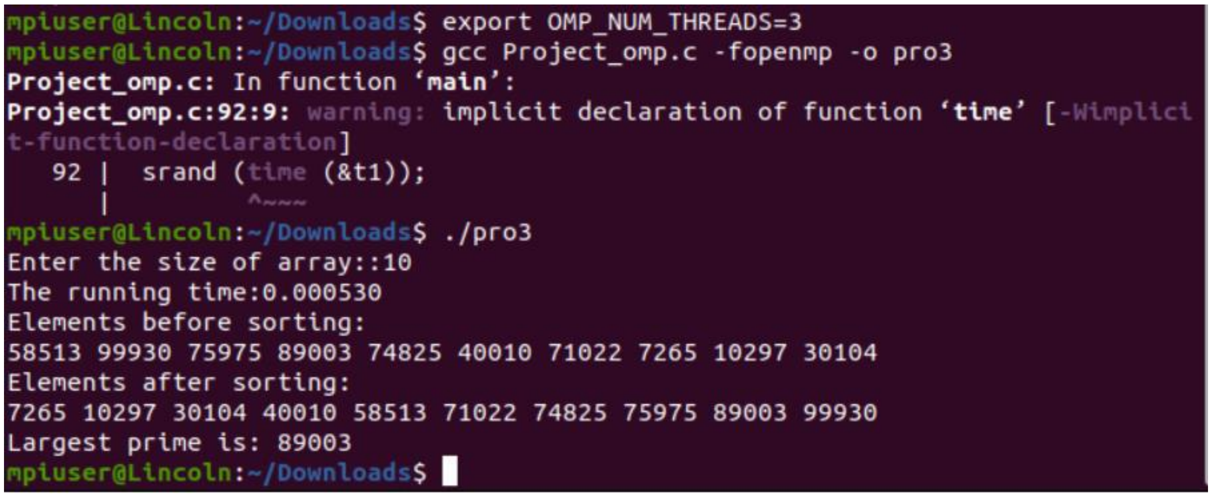
Project_omp.c results are shown below considering the size of the elements as 10 for testing purposes…



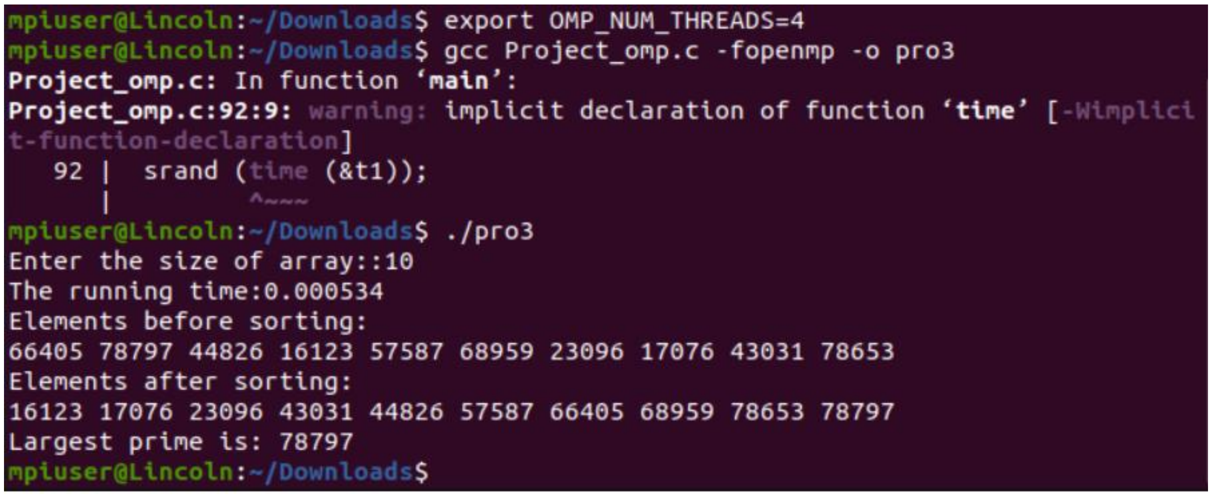
```
mpiuser@Lincoln:~/Downloads$ export OMP_NUM_THREADS=1
mpiuser@Lincoln:~/Downloads$ gcc Project_omp.c -fopenmp -o pro3
P[illegible]In function ‘main’:
P[illegible]2:9: warning: implicit declaration of function ‘time’ [-Wimplicit-function-declaration]
   92 |  srand (time (&t1));
      |         ^~~~
mpiuser@Lincoln:~/Downloads$ ./pro3
Enter the size of array::10
The running time:0.000566
Elements before sorting:
46117 53129 50274 47517 62894 37814 58016 38915 61804 36882
Elements after sorting:
36882 37814 38915 46117 47517 50274 53129 58016 61804 62894
Largest prime is: 53129
```



```
mpiuser@Lincoln:~/Downloads$ export OMP_NUM_THREADS=2
mpiuser@Lincoln:~/Downloads$ gcc Project_omp.c -fopenmp -o pro3
Project_omp.c: In function ‘main’:
Project_omp.c:92:9: warning: implicit declaration of function ‘time’ [-Wimplicit-function-declaration]
   92 |  srand (time (&t1));
      |         ^~~~
mpiuser@Lincoln:~/Downloads$ ./pro3
Enter the size of array::10
The running time:0.000520
Elements before sorting:
70788 21561 71267 51838 84834 19337 42287 11603 94539 47014
Elements after sorting:
11603 19337 21561 42287 47014 51838 70788 71267 84834 94539
No prime in the random list
```



```
mpiuser@Lincoln:~/Downloads$ export OMP_NUM_THREADS=3
mpiuser@Lincoln:~/Downloads$ gcc Project_omp.c -fopenmp -o pro3
Project_omp.c: In function ‘main’:
Project_omp.c:92:9: warning: implicit declaration of function ‘time’ [-Wimplicit-function-declaration]
   92 |  srand (time (&t1));
      |         ^~~~
mpiuser@Lincoln:~/Downloads$ ./pro3
Enter the size of array::10
The running time:0.000530
Elements before sorting:
58513 99930 75975 89003 74825 40010 71022 7265 10297 30104
Elements after sorting:
7265 10297 30104 40010 58513 71022 74825 75975 89003 99930
Largest prime is: 89003
mpiuser@Lincoln:~/Downloads$
```


```
mpiuser@Lincoln:~/Downloads$ export OMP_NUM_THREADS=4
mpiuser@Lincoln:~/Downloads$ gcc Project_omp.c -fopenmp -o pro3
Project_omp.c: In function ‘main’:
Project_omp.c:92:9: warning: implicit declaration of function ‘time’ [-Wimplici
t-function-declaration]
   92 |  srand (time (&t1));
      |         ^~~~
mpiuser@Lincoln:~/Downloads$ ./pro3
Enter the size of array::10
The running time:0.000534
Elements before sorting:
66405 78797 44826 16123 57587 68959 23096 17076 43031 78653
Elements after sorting:
16123 17076 23096 43031 44826 57587 66405 68959 78653 78797
Largest prime is: 78797
mpiuser@Lincoln:~/Downloads$
```

Benchmark:

Problem	Np=1	Np=2	Np=3	Np=4
Project_Sequential.c	0.006242	NA	NA	NA
Program_MPI.c	0.003811	0.002167	0.001793	0.000793
Project_openmp.c	0.000566	0.000520	0.000530	0.000534

Outcome and learnings:

- As mentioned earlier in our aim, we wanted to achieve improvements in execution time from all the three programs which we showed through results.
- We learnt how computation power can be divided between different resources for faster processing and problem solving through new concepts such as MPI and openmp with this project and course.