Name : Bassam Arshad

Student ID - 0259149

Summary:

1) I have written the two functions prime(long int n) and twin(long int n), by using the same logic as provided . The long int takes a number upto `2,147,483,647.` They return the nearest prime value q (q>n) to n and the nearest twin prime pair near to n, respectively. Enter the number from command line.

2) prime (long int n) and twin(long int n), are called on 4 process (0,1,2,3) separately. And utilize MPI_Send() , to send the results to process 4

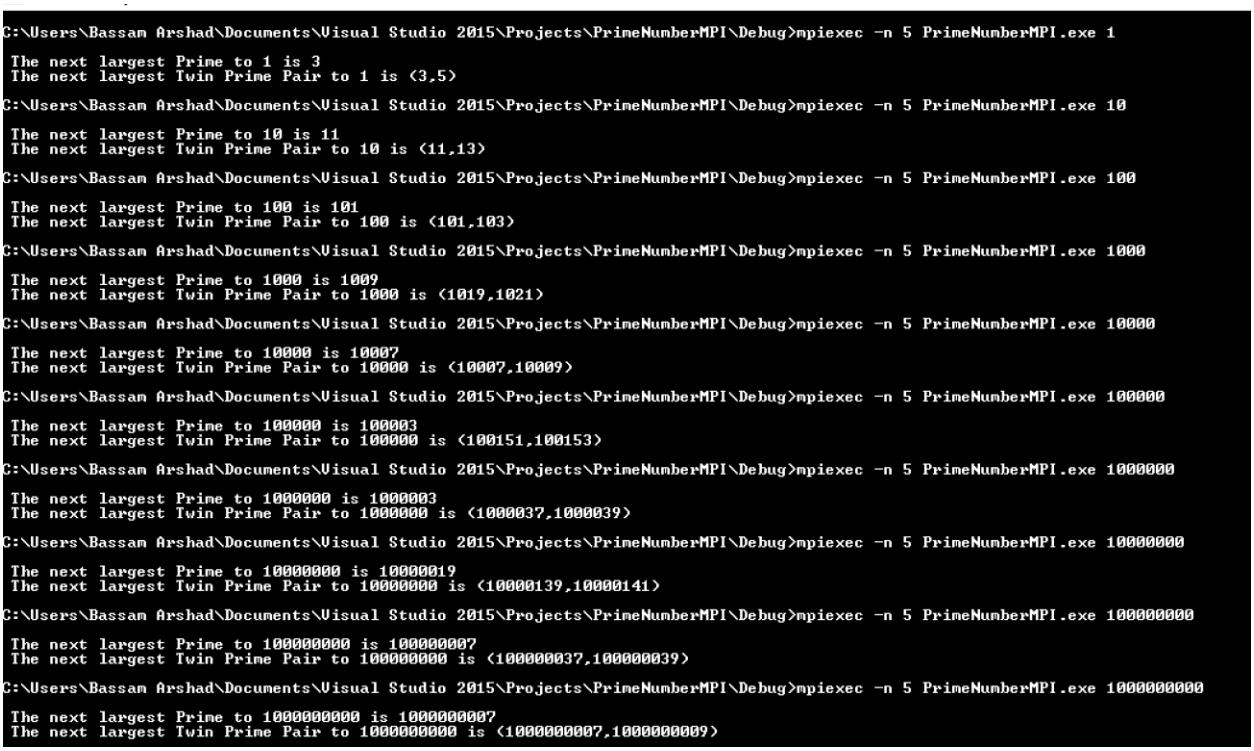
3) On process 4, we get 4 results (utilize MPI_Recv())and then we get the nearest (smallest) result. This is done using the MPI_Reduce() , function.

4) I run the program from the command line and specify the creation of 5 processes, also passing the number as argument . Using below :

   From directory of program exe file:
   mpiexec –n 5 PrimeNumberMPI.exe 10

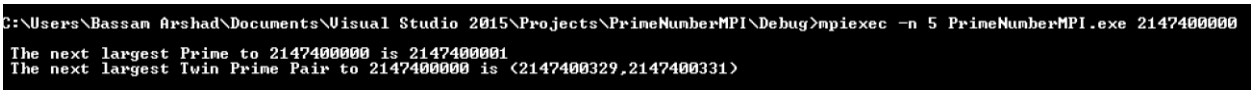
Screenshots of Results:

1,10,100,1000 to 1000000000 (1 billion)



Largest Number tested:

2147400000



C++ code:

```
/*

Date : 26th Nov, 2015
Author - Bassam Arshad
As part of CSCI 6356-01 Fall 2015 - Parallel Computer
Professor - Dr. Bin Fu

This program utilizes the MPI library to implement a parallel implementation, for
calculating the nearest (next largest) prime number for a given number
and also the nearest twin prime pair for the given number.


*/


#include <cmath>
#include <stdio.h>
#include "mpi.h"

//Function for fast prime checking for long int's
bool isPrime(long int num)
{
	if (num <= 1)
		return false;
	else if (num == 2)
		return true;
	else if (num % 2 == 0)
		return false;
	else
	{
		bool prime = true;
		long int divisor = 3;
		double num_d = static_cast<double>(num);
		long int upperLimit = static_cast<long int>(sqrt(num_d) + 1);

		while (divisor <= upperLimit)
		{
			if (num % divisor == 0)
				prime = false;
			divisor += 2;
		}
		return prime;
	}
}

//Standard logic funtion for prime checking
bool is_Prime(long int n)
{
	long int i, count = 0;
	if (n == 1 || n == 2)
		return true;
	if (n % 2 == 0)
		return false;
	for (i = 1; i <= n; i++)
	{
```

```
                if (n%i == 0)
                        count++;
        }
        if (count == 2)
                return true;
        else
                return false;
}


//Function to calculate the next largest prime number to "n"
long int prime(long int n, long int processRank)
{
        long int m = abs(n / 8);
        long int q = 1;
        bool flag = true;

        while (flag)
        {
                q = (8 * m) + (2 * processRank) + 1;
                if (isPrime(q) && (q > n))
                        //break;
                        flag = false;
                else
                        m++;
        }

        return q;
}

//Function to calculate the next largest twin prime number pair to "n"
long int twin(long int n, long int processRank)
{
        long int m = abs(n / 8);
        long int q = 1;
        bool flag = true;

        while (flag)
        {
                q = (8 * m) + (2 * processRank) + 1;
                if (isPrime(q) && isPrime((q + 2)) && (q > n))
                        flag = false;
                else
                        m++;
        }
        return q;
}

int main(int argc, char *argv[])
{

        int npes, pRank;
        long int n;

        //Range of long int  positive : 2,147,483,647
        //Getting the input from the Command line
        // Eg; Enter input number "10" like this :: mpiexec -n 5 PrimeNumberMPI.exe 10
        n = atol(argv[1]);
```

```
	//Initialize MPI Lib
	MPI_Init(&argc, &argv);
	//Total Number of Process
	MPI_Comm_size(MPI_COMM_WORLD, &npes);
	//Process Rank
	MPI_Comm_rank(MPI_COMM_WORLD, &pRank);

	MPI_Status s;

	long int smallestPrime, smallestPrimeF1;
	long int smallestPrimePair, smallestPrimePairF1;

	// MPI_Send( void *buf, int count, MPI_Datatype datatype, int dest,int tag,
MPI_Comm comm )
	// MPI_Reduce(const void *sendbuf, void *recvbuf, int count, MPI_Datatype
datatype, MPI_Op op, int root, MPI_Comm comm)

	//Process 0 computes the smallest prime and stores it in smallestPrime , then it
uses MPI_send to send the result to process 4 (with tag 0) , at the same time the
smallestPrime value
	// generated by process 0 is passed through the MPI_Reduce function at the i/p
buffer and it then sends it to process 4 (the reduce gets us a min value, across the diff
processes,
	// by using the parameter MPI_MIN, and saves the result in smallestPrimeF1)

	// the similar approach is then used for calculating the smallest twin prime pair
i.e. MPI_send along with Reduce , sending the results to process 4 and reducing the
results and saving the result in process 4.


	// The below "task allocation" can also be implemented (probably more efficiently)
in a for loop , from pRank=0 to 3 , but i am individually allocating the same for clarity
purposes ...

	if (pRank == 0)
	{
		smallestPrime = prime(n, pRank);
		MPI_Send(&smallestPrime, 1, MPI_INT, 4, 0, MPI_COMM_WORLD);
		MPI_Reduce(&smallestPrime, &smallestPrimeF1, 1, MPI_INT, MPI_MIN, 4,
MPI_COMM_WORLD);

		smallestPrimePair = twin(n, pRank);
		MPI_Send(&smallestPrimePair, 1, MPI_INT, 4, 1, MPI_COMM_WORLD);
		MPI_Reduce(&smallestPrimePair, &smallestPrimePairF1, 1, MPI_INT, MPI_MIN,
4, MPI_COMM_WORLD);
	}

	else if (pRank == 1)
	{
		smallestPrime = prime(n, pRank);
		MPI_Send(&smallestPrime, 1, MPI_INT, 4, 0, MPI_COMM_WORLD);
		MPI_Reduce(&smallestPrime, &smallestPrimeF1, 1, MPI_INT, MPI_MIN, 4,
MPI_COMM_WORLD);

		smallestPrimePair = twin(n, pRank);
		MPI_Send(&smallestPrimePair, 1, MPI_INT, 4, 1, MPI_COMM_WORLD);
```

```
		MPI_Reduce(&smallestPrimePair, &smallestPrimePairF1, 1, MPI_INT, MPI_MIN,
4, MPI_COMM_WORLD);

	}
	else if (pRank == 2)
	{
		smallestPrime = prime(n, pRank);
		MPI_Send(&smallestPrime, 1, MPI_INT, 4, 0, MPI_COMM_WORLD);
		MPI_Reduce(&smallestPrime, &smallestPrimeF1, 1, MPI_INT, MPI_MIN, 4,
MPI_COMM_WORLD);

		smallestPrimePair = twin(n, pRank);
		MPI_Send(&smallestPrimePair, 1, MPI_INT, 4, 1, MPI_COMM_WORLD);
		MPI_Reduce(&smallestPrimePair, &smallestPrimePairF1, 1, MPI_INT, MPI_MIN,
4, MPI_COMM_WORLD);
	}
	else if (pRank == 3)
	{
		smallestPrime = prime(n, pRank);
		MPI_Send(&smallestPrime, 1, MPI_INT, 4, 0, MPI_COMM_WORLD);
		MPI_Reduce(&smallestPrime, &smallestPrimeF1, 1, MPI_INT, MPI_MIN, 4,
MPI_COMM_WORLD);

		smallestPrimePair = twin(n, pRank);
		MPI_Send(&smallestPrimePair, 1, MPI_INT, 4, 1, MPI_COMM_WORLD);
		MPI_Reduce(&smallestPrimePair, &smallestPrimePairF1, 1, MPI_INT, MPI_MIN,
4, MPI_COMM_WORLD);
	}

	//  MPI_Recv(void *buf, int count, MPI_Datatype datatype, int source, int tag,
MPI_Comm comm, MPI_Status *status)

	// Process 4 recieves the smallestPrime and smallestPrimePair results from process
0,1,2,3 . It also uses the MPI_Reduce function (with MPI_MIN function) to return us the
smallest (nearest) prime to n
	// and the smallest twin prime pair to n

	else if (pRank == 4)
	{
		MPI_Recv(&smallestPrime, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, &s);
		MPI_Recv(&smallestPrimePair, 1, MPI_INT, 0, 1, MPI_COMM_WORLD, &s);
		//MPI_Reduce(&smallestPrime, &smallestPrimeF1, 1, MPI_INT, MPI_MIN, 0,
MPI_COMM_WORLD);

		MPI_Recv(&smallestPrime, 1, MPI_INT, 1, 0, MPI_COMM_WORLD, &s);
		MPI_Recv(&smallestPrimePair, 1, MPI_INT, 1, 1, MPI_COMM_WORLD, &s);
		//MPI_Reduce(&smallestPrime, &smallestPrimeF1, 1, MPI_INT, MPI_MIN, 1,
MPI_COMM_WORLD);

		MPI_Recv(&smallestPrime, 1, MPI_INT, 2, 0, MPI_COMM_WORLD, &s);
		MPI_Recv(&smallestPrimePair, 1, MPI_INT, 2, 1, MPI_COMM_WORLD, &s);
		//MPI_Reduce(&smallestPrime, &smallestPrimeF1, 1, MPI_INT, MPI_MIN, 2,
MPI_COMM_WORLD);

		MPI_Recv(&smallestPrime, 1, MPI_INT, 3, 0, MPI_COMM_WORLD, &s);
		MPI_Recv(&smallestPrimePair, 1, MPI_INT, 3, 1, MPI_COMM_WORLD, &s);
		//MPI_Reduce(&smallestPrime, &smallestPrimeF1, 1, MPI_INT, MPI_MIN, 3,
MPI_COMM_WORLD);
```

```
            MPI_Reduce(&smallestPrime, &smallestPrimeF1, 1, MPI_INT, MPI_MIN, 4,
MPI_COMM_WORLD);
            MPI_Reduce(&smallestPrimePair, &smallestPrimePairF1, 1, MPI_INT, MPI_MIN,
4, MPI_COMM_WORLD);

            printf("\n The next largest Prime to %d is %d \n", n, smallestPrimeF1);
            printf(" The next largest Twin Prime Pair to %d is (%d,%d) \n", n,
smallestPrimePairF1, smallestPrimePairF1 + 2);
      }


      MPI_Finalize();

      return 0;
      }
```