PARALLEL AND DISTRIBUTED COMPUTING
L – 19,20
DATE: 10.10.19
FACULTY : PROF. GAYATHRI R.
DHRUBANKA DUTTA, 17BCE1019

Seive of Eratosthenes

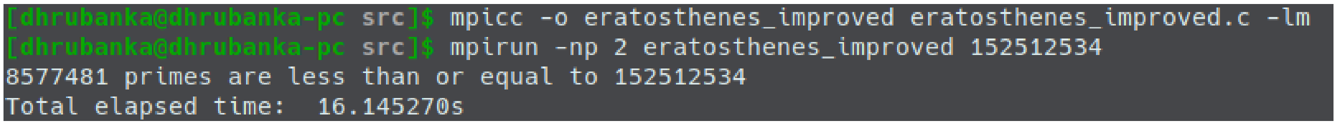
CODE :

```
#include <mpi.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include "MyMPI.h"


int main(int argc, char** argv)                    {
    int     count;                /* local prime count */
    double  elapsed_time;         /* parallel execution time */
    int     first;                /* index of first multiple */
    int     global_count;         /* global prime count */
    int     high_value;           /* highest value on this proc */
    int     i;
    int     id;                   /* process id number */
    int     index;                /* index of current prime */
    int     low_value;            /* lowest value on this proc */
    int     n;                    /* sieving from 2, ..., n */
    int     p;                    /* number of processes */
    int     proc0_size;           /* size of proc 0's subarray */
    int     prime;                /* current prime */
    int     size;                 /* elements in marked string */
    int     first_value_index;
    int     prime_step;
    int     prime_doubled;
    int     sqrt_n;
    int     prime_multiple;
    int     num_per_block;
    int     block_low_value;
    int     block_high_value;
    int     first_index_in_block;
    char*   marked;               /* portion of 2, ..., n */
    char*   primes;

    MPI_Init(&argc, &argv);

    /* start the timer */
    MPI_Barrier(MPI_COMM_WORLD);
    elapsed_time = -MPI_Wtime();
```

```
MPI_Comm_rank(MPI_COMM_WORLD, &id);
MPI_Comm_size(MPI_COMM_WORLD, &p);

if (argc != 2)    {
    if (id == 0) /* parent process */
        printf("Command line: %s <m>\n", argv[0]);
    MPI_Finalize();
    exit(1);
} /* if (argc != 2) */

n = atoi(argv[1]);

/*
 * Figure out this process's share of the array, as well as the
 * integers represented by the first and last array elements
 */
low_value  = BLOCK_FIRST + BLOCK_LOW(id, p, n - 1)  * BLOCK_STEP;
high_value = BLOCK_FIRST + BLOCK_HIGH(id, p, n - 1) * BLOCK_STEP;
size       = BLOCK_SIZE(id, p, n - 1);

/*
 * bail out if all the primes used for sieving are not all
 * help by process 0
 */
proc0_size = (n - 1) / p;

if ((2 + proc0_size) < (int)sqrt((double)n))    {
    if (id == 0) /* parent process */
        printf("Too many processes\n");
    MPI_Finalize();
    exit(1);
} /* if */

// compute primes from 2 to sqrt(n);
sqrt_n = sqrt(n);
primes = (char*)calloc(sqrt_n + 1, 1);
for (prime_multiple = 2;
     prime_multiple <= sqrt_n;
     prime_multiple += 2)    {
    primes[prime_multiple] = 1;
} /* for */

for (prime = 3; prime <= sqrt_n; prime += 2)    {
    if (primes[prime] == 1)
        continue;

    for (prime_multiple = prime << 1;
         prime_multiple <= sqrt_n;
         prime_multiple += prime)    {
        primes[prime_multiple] = 1;
    }
} /* for */

/*
 * allocate this process' share of the array
 */
marked = (char*)calloc(size * sizeof(char), 1);
if (marked == NULL)    {
    printf("Cannot allocate enough memory\n");
    MPI_Finalize();
    exit(1);
} /* if */

num_per_block    = 1024 * 1024;
block_low_value  = low_value;
block_high_value = MIN(high_value,
                       low_value + num_per_block * BLOCK_STEP);

for (first_index_in_block = 0;
     first_index_in_block < size;
     first_index_in_block += num_per_block)    {
    for (prime = 3; prime <= sqrt_n; prime++)       {
        if (primes[prime] == 1)
            continue;
        if (prime * prime > block_low_value)   {
            first = prime * prime;
        }
       else   {
            if (!(block_low_value % prime))    {
                first = block_low_value;
            }
            else    {
                first = prime - (block_low_value % prime) +
                        block_low_value;
            }
       }

       /*
        * optimization - consider only odd multiples
        *                 of the prime number
        */
       if ((first + prime) & 1) // is odd
          first += prime;

       first_value_index = (first - BLOCK_FIRST) / BLOCK_STEP -
                           BLOCK_LOW(id, p, n - 1);
       prime_doubled     = prime << 1;
```

```
            prime_step        = prime_doubled / BLOCK_STEP;
            for (i = first; i <= high_value; i += prime_doubled)   {
                marked[first_value_index] = 1;
                first_value_index += prime_step;
            } /* for */
        }

        block_low_value += num_per_block * BLOCK_STEP;
        block_high_value = MIN(high_value,
                           block_high_value + num_per_block * BLOCK_STEP);
    } /* for first_index_in_block */


    /*
     * count the number of prime numbers found on this process
     */
    count = 0;
    for (i = 0; i < size; i++)
        if (!marked[i])
            count++;

    MPI_Reduce(&count, &global_count, 1, MPI_INT,
               MPI_SUM, 0, MPI_COMM_WORLD);

    /*
     * stop the timer
     */
    elapsed_time += MPI_Wtime();

    /* print the results */
    if (id == 0)   {
        global_count += 1; /* add first prime, 2 */
        printf("%d primes are less than or equal to %d\n",
               global_count, n);
        printf("Total elapsed time: %10.6fs\n",
               elapsed_time);
    } /* if */

    MPI_Finalize();

    return 0;
}
```

OUTPUT :



```
[dhrubanka@dhrubanka-pc src]$ mpicc -o eratosthenes_improved eratosthenes_improved.c -lm
[dhrubanka@dhrubanka-pc src]$ mpirun -np 2 eratosthenes_improved 152512534
8577481 primes are less than or equal to 152512534
Total elapsed time:  16.145270s
```