**PRACTICAL # 08**

**OBJECT:**

Counting in parallel the number of primes between 1 and N, using OpenMPI.

**THEORY:**

One of the important challenges of number theory is dealing with the prime numbers. The prime numbers have important applications in cryptography and digital data security. Finding large prime numbers is computationally complex task. This complexity can be reduced using the parallel processing.

The program below finds the prime numbers between a given ranges of numbers. For each integer **i**, it simply checks whether any smaller j evenly divides it. The total amount of work for a given N is thus roughly proportional to 1/2\*N^2.

**Code:**

The program here uses new message passing functions.

MPI::COMM\_WORLD.Bcast ( &n, 1, MPI::INT, 0 );

MPI::COMM\_WORLD.Reduce ( &primes\_part, &primes, 1, MPI::INT, MPI::SUM, 0 );

Instead of passing the message to the each process individually, the COMM\_WORLD.Bcast function broadcasts the message to all the processors.

The COMM\_WORLD.Reduce collects back all the received messages and reduces them into the primes.

***# include*** *<cstdlib>*

***# include*** *<iostream>*

***# include*** *<iomanip>*

***# include*** *<cmath>*

***# include*** *<ctime>*

***# include*** *<mpi.h>*

***# define*** *TIME\_SIZE 20*

***using******namespace*** *std;*

***int******prime\_number*** *(* ***int*** *n,* ***int*** *id,* ***int*** *p );*

***void******timestamp*** *( );*

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

*{*

***int*** *i, id;*

***int*** *n, n\_factor, n\_hi, n\_lo;*

***int*** *p, primes, primes\_part;*

***double*** *wtime;*

*n\_lo = 1;*

*n\_hi = 10000;*

*n\_factor = 2;*

*//Initialize MPI*

*MPI::Init ( argc, argv );*

*// Get the number of processes*

*p = MPI::COMM\_WORLD.Get\_size ( );*

*// Determine this processes's rank*

*id = MPI::COMM\_WORLD.Get\_rank ( );*

***if*** *( id == 0 )*

*{*

*timestamp ( );*

*cout << " \n MPI program to count the number of primes.\n";*

*cout << " The number of processes is " << p << "\n";*

*cout << " \n Num Prime Count Time\n\n";*

*}*

*n = n\_lo;*

***while*** *( n <= n\_hi )*

*{*

***if*** *( id == 0 )*

*{*

*wtime = MPI::Wtime ( );*

*}*

*MPI::COMM\_WORLD.Bcast ( &n, 1, MPI::INT, 0 );*

*primes\_part = prime\_number ( n, id, p );*

*MPI::COMM\_WORLD.Reduce ( &primes\_part, &primes, 1, MPI::INT, MPI::SUM, 0 );*

***if*** *( id == 0 )*

*{*

*wtime = MPI::Wtime ( ) - wtime;*

*cout << " " << setw(8) << n*

*<< " " << setw(10) << primes*

*<< " " << setw(28) << wtime << "\n";*

*}*

*n = n \* n\_factor;*

*}*

*// Terminate MPI.*

*MPI::Finalize ( );*

***if*** *( id == 0 )*

*{*

*cout << "\n PRIME\_MPI - Master process:\n"<< " Normal end of execution.\n";*

*timestamp ( );*

*}*

***return*** *0;*

*}*

***int******prime\_number*** *(* ***int*** *n,* ***int*** *id,* ***int*** *p )*

*// PRIME\_NUMBER returns the number of primes between 1 and N.*

*// Parameters:*

*// Inputs: int N, the maximum number to check. int ID, the ID of this process, between 0 and P-1. int P, the number of processes.*

*// Output, int PRIME\_NUMBER, the number of prime numbers up to N.*

*{*

***int*** *i, j;*

***int*** *prime, total;*

*total = 0;*

***for*** *( i = 2 + id; i <= n; i = i + p )*

*{*

*prime = 1;*

***for*** *( j = 2; j < i; j++ )*

*{*

***if*** *( ( i % j ) == 0 )*

*{*

*prime = 0;*

***break****;*

*}*

*}*

*total = total + prime;*

*}*

***return*** *total;*

*}*

***void******timestamp*** *( )*

*// TIMESTAMP prints the current YMDHMS date as a time stamp.*

*{*

***static******char*** *time\_buffer[TIME\_SIZE];*

***const******struct*** *tm \*tm;*

*size\_t len;*

*time\_t now;*

*now =* ***time*** *( NULL );*

*tm =* ***localtime*** *( &now );*

*len =* ***strftime*** *( time\_buffer, TIME\_SIZE, "%d %B %Y %I:%M:%S %p", tm );*

*cout << time\_buffer << "\n";*

***return****;*

*}*

To compile the program type;

mpic++ primes.cpp -o primes

To run the program;

mpirun -np 3 primes

The output of the program will be of the following format.

// N PRIME\_NUMBER

//

// 1 0

// 10 4

// 100 25

// 1,000 168

// 10,000 1229

// 100,000 9592

// 1,000,000 78,498

// 10,000,000 664,579

// 100,000,000 5,761,455

// 1,000,000,000 50,847,534

**ACTIVITIES**

**Activity 1**

Execute the program given above and observe the results.

**REVIEW QUESTIONS**

1. What is the purpose of COMM\_WORLD.Bcast and COMM\_WORLD.Reduce functions?
2. What is the importance of prime numbers?
3. What is the function of master processor in a parallel computing environment?