**MPI (Message- passing Interface)**

MPI (Message Passing Interface) program to calculate the sum of prime numbers among a set of 100 random numbers.

**Justification for Selecting this program:**

MPI showcases the essence of parallel computing. It is an excellent choice to demonstrate parallel computing concepts, algorithmic complexity, scalability, data distribution, and communication among processes. Prime number calculation involves multiple iterations and can benefit from parallelization. This task allows for scalability demonstrations in parallel computing. With MPI, it's possible to distribute the workload among multiple processes efficiently, and understanding how the program scales as the number of processors increases.

**Program:**

#include <iostream>

#include <mpi.h>

#include <cmath>

bool is\_prime(int n) {

if (n <= 1) return false;

if (n == 2) return true;

if (n % 2 == 0) return false;

for (int i = 3; i <= std::sqrt(n); i += 2) {

if (n % i == 0) return false;

}

return true;

}

int main(int argc, char\*\* argv) {

MPI\_Init(&argc, &argv);

int rank; // Process rank

int size; // Total number of processes

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);

MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);

const int num\_primes = 100;

int prime\_sum = 0;

int local\_sum = 0;

// Calculate prime numbers and sum them up

for (int num = rank + 2; num <= num\_primes; num += size) {

if (is\_prime(num)) {

local\_sum += num;

}

}

// Reduce local sums to the root process to get the global sum

MPI\_Reduce(&local\_sum, &prime\_sum, 1, MPI\_INT, MPI\_SUM, 0, MPI\_COMM\_WORLD);

if (rank == 0) {

std::cout << "Sum of the first " << num\_primes << " prime numbers: " << prime\_sum << std::endl;

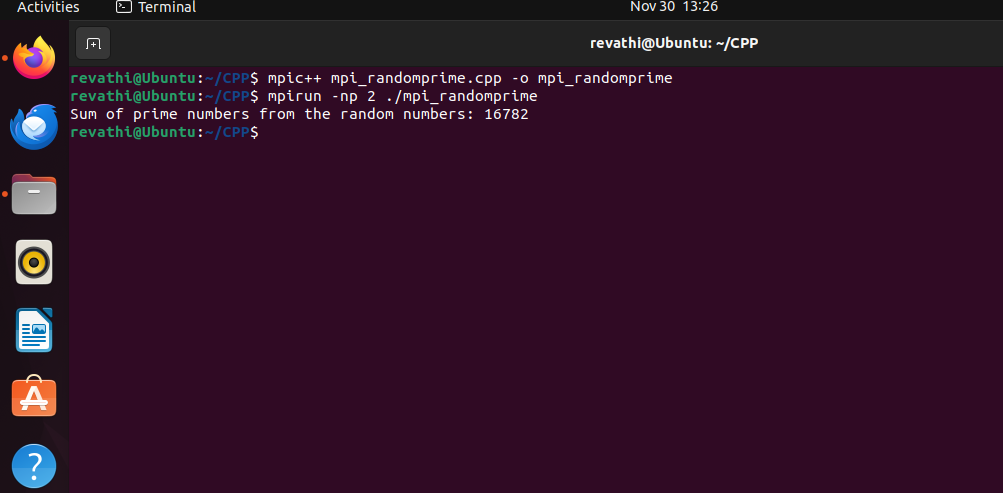
}

MPI\_Finalize();

return 0;

}

**MPI program output:**

****

**Program Explanation:**

This program uses MPI (Message Passing Interface), a tool for parallel programming, to find the sum of the first 100 prime numbers using multiple processes

**MPI Functions (MPI\_Init, MPI\_Comm\_rank, MPI\_Comm\_size, MPI\_Reduce, MPI\_Finalize): These functions are part of the MPI library. They handle initializing MPI, retrieving the rank and size of processes, performing reduction operations, and finalizing MPI operations respectively. They enable communication and coordination among different processes**

**Include Libraries**: The program includes necessary libraries (‘**iostream’** for input/output, ‘**mpi.h’** for MPI functions, and ‘**cmath’** for mathematical functions).

**Parallelization using for Loop**: The **for loop** inside the main function distributes the workload among different processes. Each process starts checking numbers from a specific point (rank + 2) up to 100, incrementing by the total number of processes (size). This division of work enables parallel computation, as each process handles a different portion of the task.

**Root Process Check (if (rank == 0)):** The conditional check ensures that only the root process (rank 0) displays the final result. This prevents multiple processes from printing the result simultaneously.

**Mathematical Operations (std::sqrt, %):** The program uses % for modulo operation to check divisibility and std::sqrt function to find the square root of a number while determining prime numbers. These operations are essential for the prime number checking algorithm.

**Function to Check Prime Numbers** (‘**is\_prime’**): There's a function named is\_prime that checks if a number is prime or not. It goes through a set of rules to determine if the number given to it is a prime number or not.

**Main Function**: The ‘**main**’ function is where the program starts its execution.

**Initializing MPI**: The program initializes MPI with ‘**MPI\_Init’**, getting the process rank (rank) and the total number of processes (‘size’).

**Variables**: The program creates variables to store the sum of prime numbers **(‘prime\_sum’**) and the local sum calculated by each process **(‘local\_sum’**).

**Prime Number Calculation**: Each process starts calculating prime numbers from a specific starting point (**rank + 2**) up to 100, with a step size determined by the total number of processes **(‘size’**). Every process checks if the numbers in its range are prime and adds the prime numbers to its local sum.

**Reduction Operation**: The program uses **‘MPI\_Reduce’** to combine all the local sums from different processes into a single value (prime\_sum) on the root process (rank 0).

**Displaying Result**: If the process is the root (rank 0), it prints the final sum of the first 100 prime numbers.

**Finalizing MPI**: The program ends its MPI operations with **‘MPI\_Finalize’**.