ACS COLLEGE OF ENGINEERING

#207, Kambipura, Mysore road, Banglore-74

Department of Computer Science & Engineering

LAB MANUAL
Academic Year 2025-26

PARALLEL COMPUTING LABORATORY

(For VII Semester BE)

SUBJECT CODE: BCS702

Sponsored by
MOOGAMBIGAI CHARITABLE & EDUCATION TRUST
BANGALORE-560074

PROGRAM - 1

Write a OpenMP program to sort an array on n elements using both sequential and parallel mergesort (using Section). Record the difference in execution time.

PROGRAM:

```
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
#include <time.h>

// ----------------  MERGE FUNCTION ----------- -----
void merge(int arr[], int left, int mid, int right) {
   int i, j, k;
   int n1 = mid - left + 1;
   int n2 = right - mid;

   int *L = (int *)malloc(n1 * sizeof(int));
   int *R = (int *)malloc(n2 * sizeof(int));

   for (i = 0; i < n1; i++)
      L[i] = arr[left + i];
   for (j = 0; j < n2; j++)
      R[j] = arr[mid + 1 + j];

   i = 0;
   j = 0;
   k = left;

   while (i < n1 && j < n2)
      arr[k++] = (L[i] <= R[j]) ? L[i++] : R[j++];

   while (i < n1)
      arr[k++] = L[i++];
   while (j < n2)
      arr[k++] = R[j++];

   free(L);
   free(R);
}
```

```
// ----------------  SERIAL MERGE SORT ----------------
void serialMergeSort(int arr[], int left, int right) {
   if (left < right) {
      int mid = left + (right - left) / 2;
      serialMergeSort(arr, left, mid);
      serialMergeSort(arr, mid + 1, right);
      merge(arr, left, mid, right);
   }
}
// ----------------  PARALLEL MERGE SORT ----------------
void parallelMergeSort(int arr[], int left, int right, int  depth) {
   if (left < right) {
      int mid = left + (right - left) / 2;
   if (depth <= 4) {
         #pragma omp parallel sections
         {
            #pragma omp section
            parallelMergeSort(arr, left, mid, depth + 1);

            #pragma omp section
            parallelMergeSort(arr, mid + 1, right, depth + 1);
         }
      } else {
         serialMergeSort(arr, left, mid);
         serialMergeSort(arr, mid + 1, right);
      }

      merge(arr, left, mid, right);
   }
}

// ----------------  MAIN FUNCTION ----------------
int main() {
   int n;
   printf("Enter the number of elements in the array: ");
   scanf("%d", &n);

   int *arr_seq = (int *)malloc(n * sizeof(int));
   int *a rr_par = (int *)malloc(n * sizeof(int));
   if (!arr_seq || !arr_par) {
      printf("Memory allocation failed. \n");
      return -1;
   }
```

```
srand(time(NULL));
   for (int i = 0; i < n; i++) {
      int val = rand() % 100000;
      arr_seq[i]  = val;
      arr_par[i] = val;
   }

   double start, end;


    // Serial sort
   start = omp_get_wtime();
   serialMergeSort(arr_seq, 0, n - 1);
   end = omp_get_wtime();
   double time_serial = end - start;

   // Parallel sort
   start = omp_get_wtime();
   parallelMergeSort(arr_par, 0, n - 1, 0);
   end = omp_get_wtime();
   double time_parallel = end - start;

   printf(" \nSerial Merge Sort Time   : %.6f seconds \n", time_serial);
   printf("Parallel Merge Sort Time : %.6f seconds \n", time_parallel);
   printf("Speedup                  : %.2f× \n", time_serial / time_parallel);

   // Optional: Verify correctness
   for (int i = 0; i < n; i++) {
      if (arr_seq[i] != arr_par[i]) {
         printf("Mismatch at index %d!\n", i);
         break;
      }
   }

   free(arr_seq);
   free(arr_par);
   return 0;
}
```

SAMPLE OUTPUT:

```
. /opt/Xilinx/14.7/ISE_DS/ISE/.settings64.sh /opt/Xilinx/14.7/ISE_DS/ISE
gcem@gcem-OptiPlex-Tower-7010:~$ gcc -fopenmp pgm1.c -o pgm1
gcem@gcem-OptiPlex-Tower-7010:~$ ./pgm1
Enter the number of elements in the array: 10000

Serial Merge Sort Time   : 0.004651 seconds
Parallel Merge Sort Time : 0.001802 seconds
Speedup                  : 2.58×
gcem@gcem-OptiPlex-Tower-7010:~$
```

PROGRAM - 2

Write an OpenMP program that divides the Iterations into chunks containing 2 iterations, respectively (OMP_SCHEDULE=static,2). Its input should be the number of iterations, and its output should be which iterations of a parallelized for loop are executed by which thread.
For example, if there are two threads and four iterations, the output might be the following:
c. Thread 0 : Iterations 0 −− 1
d. Thread 1 : Iterations 2 −− 3

PROGRAM:

```
#include <stdio.h>
#include <omp.h>
int main() {
   int num_tasks, num_threads;
   // Get user input
   printf("Enter the number of tasks: ");
   scanf("%d", &num_tasks);
   printf("Enter the number of threads: ");
   scanf("%d", &num_threads);
   printf(" ------------------------------------ \n");

   // Set number of threads
   omp_set_num_threads(num_threads);

   // Parallel for loop with static scheduling and chunk size 2
   #pragma omp parallel for schedule(static, 2)
   for (int i = 0; i < num_tasks; i++) {

       int tid = omp_get_thread_num();

      // Critical section to prevent mixed output
      #pragm a omp critical
      {
         printf("Thread %d executes iteration %d \n", tid, i);
      }
   }
   return 0;
}
```

SAMPL E OUTPUT:

```
Thread 2 executes iteration 4
Thread 1 executes iteration 3
Thread 2 executes iteration 5
gcem@gcem-OptiPlex-Tower-7010:~$ gcc -fopenmp pg2mod.c -o pg2mod
gcem@gcem-OptiPlex-Tower-7010:~$ ./pg2mod
Enter the number of tasks: 24
Enter the number of threads: 12
-------------------------------------
Thread 0 executes iteration 0
Thread 0 executes iteration 1
Thread 11 executes iteration 22
Thread 1 executes iteration 2
Thread 7 executes iteration 14
Thread 4 executes iteration 8
Thread 10 executes iteration 20
Thread 7 executes iteration 15
Thread 6 executes iteration 12
Thread 6 executes iteration 13
Thread 5 executes iteration 10
Thread 5 executes iteration 11
Thread 1 executes iteration 3
Thread 8 executes iteration 16
Thread 8 executes iteration 17
Thread 11 executes iteration 23
```

PROGRAM - 3

Write a OpenMP program to calculate n Fibonacci numbers using tasks.

PROGRAM:

```
#include <stdio.h>
#include <omp.h>
// Recursive Fibonacci function with OpenMP tasks
int fib(int n) {
   if (n < 2)
      return n;
   int x, y;
  #pragma omp task shared(x)
   x = fib(n - 1);
   #pragma omp t ask shared(y)
   y = fib(n - 2);
   #pragma omp taskwait
   return x + y;
}
int main() {
   int n;
   printf("Enter the number of Fibonacci terms: ");
   scanf("%d", &n);
    printf("Fibonacci Series: \n");
   #pragma omp parallel
   {
      #pragma omp single
      {
         for (int i = 0; i < n; i++) {
            #pragma omp task firstprivate(i)
            {
               int result = fib(i);
               #pragma omp critical
               printf("Fib(%d) = %d \n", i, result);
            }
         }
      }
   }
   return 0;
}
```

SAMPLE OUTPUT:

PROGRAM - 4

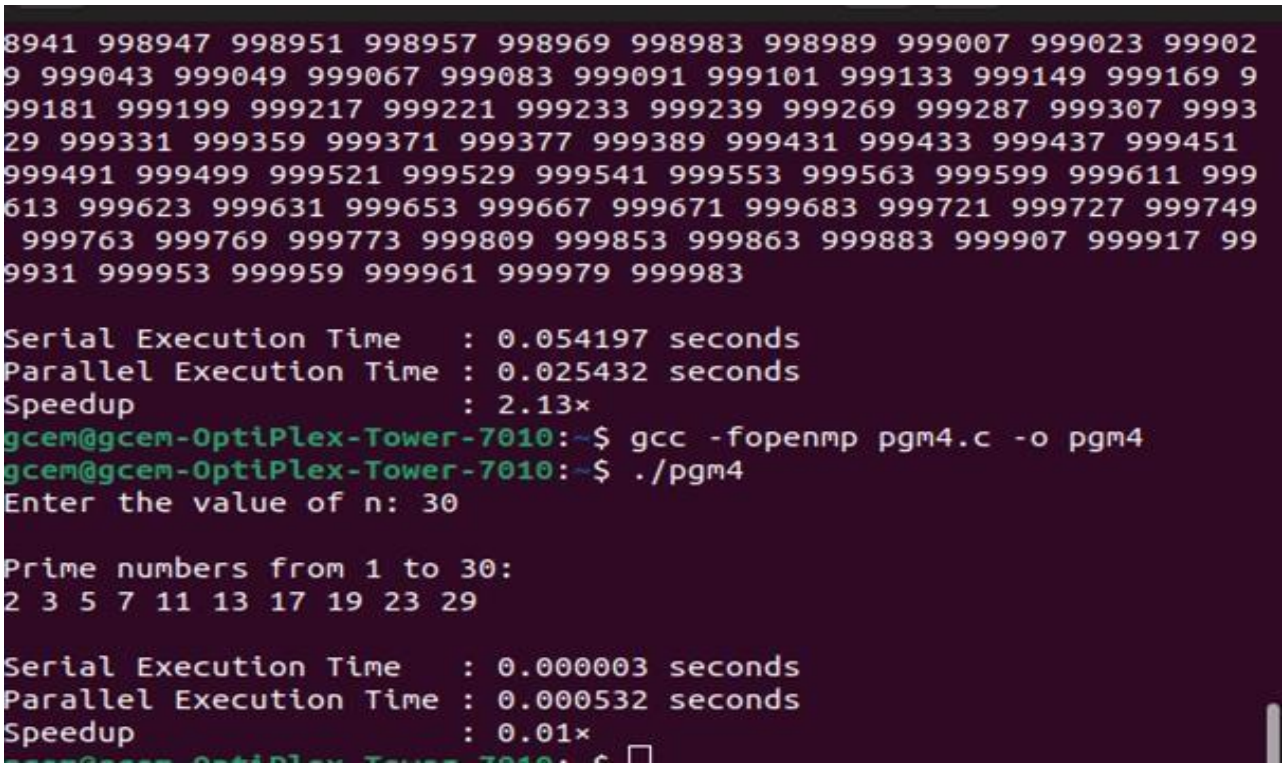
Write a OpenMP program to find the prime numbers from 1 to n employing parallel for directive. Record both serial and parallel execution times.

PROGRAM:

```
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
#include <time.h>
// Function to check if a number is prime
int is_prime(int num) {
    if (num <= 1) return 0;
    if (num == 2) return 1;
    if (num % 2 == 0) return 0;
    for (int i = 3; i * i <= num; i += 2)
        if (num % i == 0)
            return 0;
    return 1;
}
int main() {
    int n;
   printf("Enter the value of n: ");
    scanf("%d", &n);
    int *primes_serial = (int *)malloc(n * sizeof(int));
    int *primes_parallel = (int *)malloc(n * sizeof(int));
    // ----------  SERIAL EXECUTION ----------
    double start_serial = omp_get_wti me();
    for (int i = 1; i <= n; i++) {
        if (is_prime(i)) {
            primes_serial[i - 1] = 1;
        } else {
            primes_serial[i - 1] = 0;
        }
    }
    double end_serial = omp_get_wtime();
    double time_serial = end_serial - start_serial;
    // ----------  PARALLEL EXECUTION ----------
    double start_parallel = omp_get_wtime();
```

```
 #pragma omp parallel for
    for (int i = 1; i <= n; i++) {
        if (is_prime(i)) {
            primes_parallel[i - 1] = 1;
        } else  {
            primes_parallel[i - 1] = 0;
        }
    }

    double end_parallel = omp_get_wtime();
    double time_parallel = end_parallel - start_parallel;

    // ----------  OUTPUT ----------
    printf(" \nPrime numbers from 1 to %d: \n", n);
    for (int i = 1; i <= n; i++) {
        if (primes_parallel[i - 1])
            printf("%d ", i);
    }

    printf(" \n\nSerial Execution Time   : %.6f seconds", time_serial);
    printf(" \nParallel Execution Time : %.6f seconds", time_parallel);
    printf(" \nSpeedup                 : %.2f× \n", time_serial / time_parallel);

    free(primes_serial);
    free(primes_parallel);

    return 0;

}
```

SAMPLE OUTPUT:



PROGRAM - 5

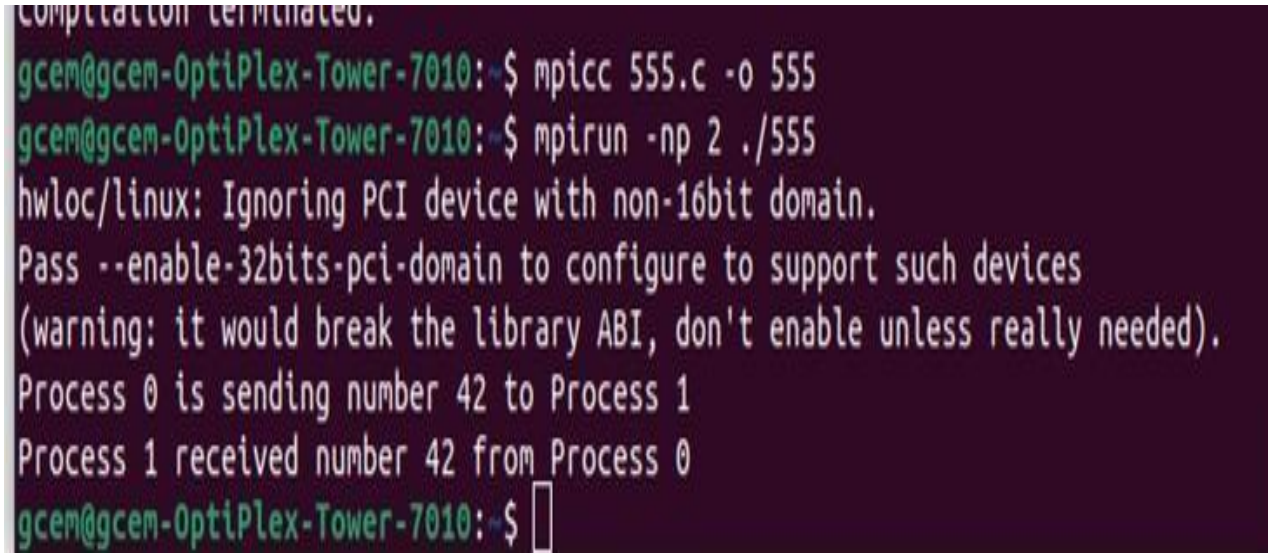
Write a MPI Program to demonstration of MPI_Send and MPI_Recv.

PROGRAM:

```
// Filename: mpi_send_recv_demo.c
#include <stdio.h>
#include <mpi.h>
int main(int argc, char *argv[])
{
   int rank, size;
   int number;

   // Initialize the MPI environment
   MPI_Init(&argc, &argv);

   // Get the number of processes
   MPI_Comm_size(MPI_COMM_WORLD, &size);

   // Get the rank of the process
   MPI_Comm_rank(MPI_COMM_WORLD, &rank);

   if (size < 2)
   {
      if (rank == 0)
      {
         printf ("This program requires at least 2 processes. \n");
      }
      MPI_Finalize();
      return 0;
   }
   if (rank == 0)
   {
      // Process 0 sends a number to Process 1
      number = 42;
      printf("Process 0 is sending number %d to Process 1 \n", number);
      MPI_Send(&number, 1, MPI_INT, 1, 0, MPI_COMM_WORLD);
   }
```

```
  else if (rank == 1)
    {


  // Process 1 receives a number from Process 0
        MPI_Recv(&number, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        printf("Process 1 received number %d from Process 0 \n", number);
    }

    // Finalize the MPI environment
    MPI_Finalize();
    return 0;
}
```

SAMPLE OUTPUT:



PROGRAM - 6

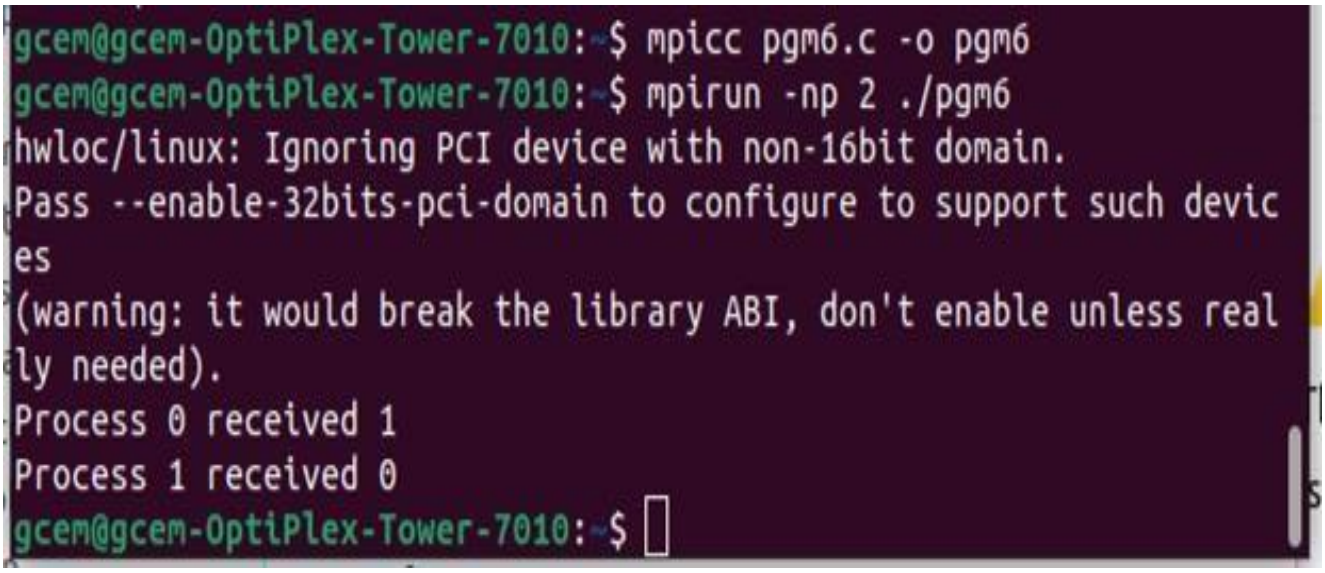
Write a MPI program to demonstration of deadlock using point to point com munication and avoidance of deadlock by altering the call sequence

PROGRAM :

```
// deadlock_mpi.c
#include <stdio.h>
#include <mpi.h>

int main(int argc, char *argv[])
{
   int rank, data_send, data_recv;

   MPI_Init(&argc, &argv);
   MPI_Comm_rank(MPI_COMM_WORLD,
&rank);

   data_send = rank;

   if (rank == 0)
   {
      MPI_Send(&data_send, 1, MPI_INT, 1, 0,
MPI_COMM_WORLD);
      MPI_Recv(&data_recv, 1, MPI_INT, 1, 0,
MPI_COMM_WORLD, MPI_STATUS_IGNORE);
   }
   else if (rank == 1)
   {
      MPI_Send(&data_send, 1, MPI_INT, 0, 0,
MPI_COMM_WORLD);
      MPI_Recv(&data_recv, 1, MPI_INT, 0, 0,
MPI_COMM_WORLD, MPI_STATUS_IGNORE);
   }
   printf("Process %d received %d \n", rank, data_recv);

    MPI_Finalize();
   return 0;
```

}

SAMPLE OUTPUT:



PROGRAM - 7

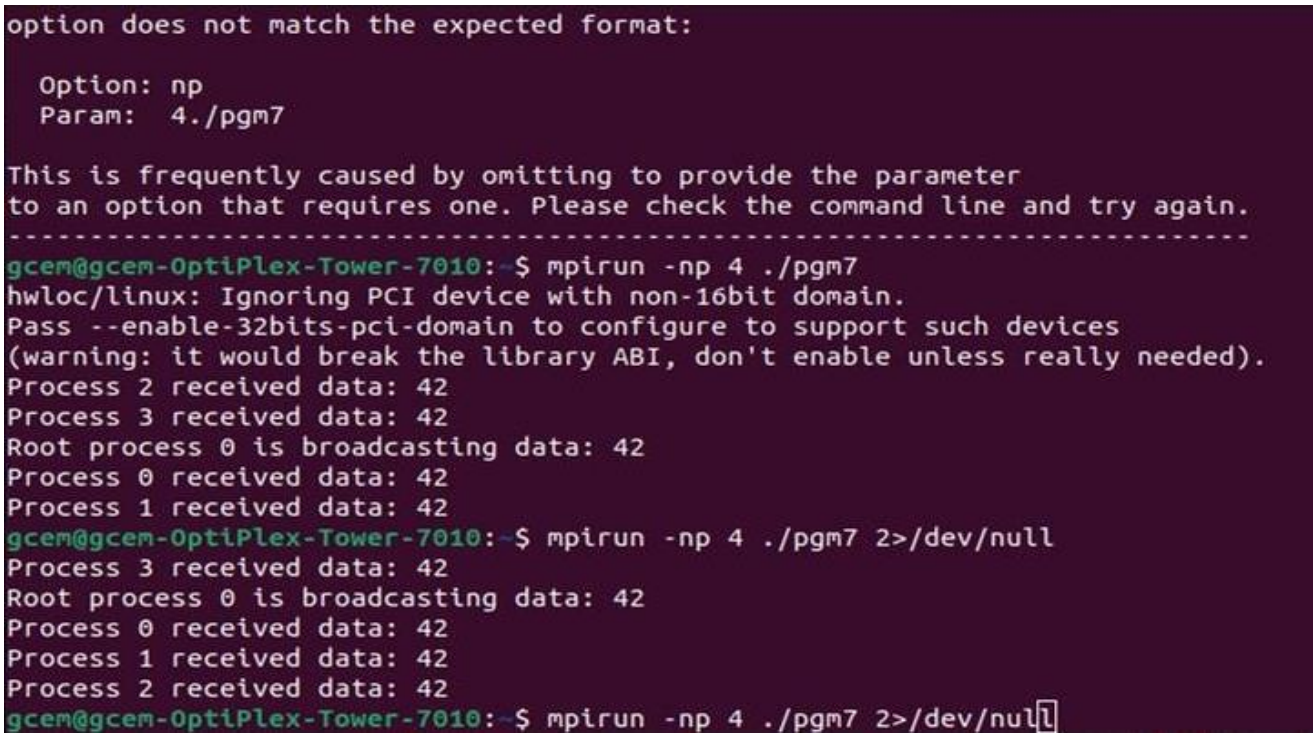
Write a MPI Program to demonstration of Broadcast operation.

PROGRAM:

```
#include <stdio.h>
#include <mpi.h>

int main(int argc, char** argv) {
    int rank, size;
   int data = 0; // The variable to broadcast

   MPI_Init(&argc, &argv);              // Initialize the
MPI environment
   MPI_Comm_rank(MPI_COMM_WORLD, &rank);
// Get the rank of the process
   MPI_Comm_size(MPI_COMM_WORLD, &size);
// Get the total number of processes

   if (rank == 0) {
      data = 42; // Only root process sets the data
      printf("Root process %d is broadcasting data:
%d\n", rank, data);
   }
   // Broadcast the value of 'data' from process 0 to all
other processes
   MPI_Bcast(&data, 1, MPI_INT, 0,
MPI_COMM_WORLD);

   // All processes print the received data
   print f("Process %d received data: %d \n", rank, data);
```

```
MPI_Finalize(); // Finalize the MPI environment
    return 0;
}
```

SAMPLE OUTPUT:



PROGRAM - 8

Write a MPI Program demonstration of MPI_Scatter and MPI_Gather

PROGRAM :

```
#include <stdio.h>
#include <mpi.h>
int main(int argc, char *argv[]) {
   int rank, size;
   int data[8];            // Array to be scattered
   int recv_data;          // Each process receives one int
   int gathered_data[8];   // Array to gather results
   MPI_Init(&argc, &argv);
   MPI_Comm_rank(MPI_COMM_WORLD, &rank);
   MPI_Comm_size(MPI_COMM_WORLD, &size);
   // Assume size is 8 or less
   if (size > 8) {
      if (rank = = 0) {
         printf("This demo supports up to 8 processes only. \n");
      }
      MPI_Finalize();
      return 0;
   }
   // Only root process initializes the data
   if (rank == 0) {
      for (int i = 0; i < size; i++) {
         data[i] = i * 10;
      }
      printf("Root process has data: ");
      for (int i = 0; i < size; i++) {
         printf("%d ", data[i]);
      }
      printf(" \n");
   }
   // Scatter data to all processes
   MPI_Scatter(data, 1, MPI_INT , &recv_data, 1, MPI_INT, 0, MPI_COMM_WORLD);

   // Each process prints its received data
   printf("Process %d received %d \n", rank, recv_data);
```

```
// Let's modify the received data (e.g., multiply by 2)
   recv_data *= 2;
   // Gather the modifie d data back to root
   MPI_Gather(&recv_data, 1, MPI_INT, gathered_data, 1, MPI_INT, 0,
MPI_COMM_WORLD);
   // Root prints gathered results
   if (rank == 0) {
      printf("Root process gathered: ");
      for (int i = 0; i < size; i++) {
         printf("%d ", gathered_data[i]);
      }
      printf(" \n");
   }

   MPI_Finalize();
   return 0;
}
```

SAMPLE OUTPUT:

```
rocess 1 received 0
cem@gcem-OptiPlex-Tower-7010:~$ mpicc pgm8.c -o pgm8
cem@gcem-OptiPlex-Tower-7010:~$ mpirun -np 4 ./pgm8
wloc/linux: Ignoring PCI device with non-16bit domain.
ass --enable-32bits-pci-domain to configure to support such devices
warning: it would break the library ABI, don't enable unless really nee
ed).
oot process has data: 0 10 20 30
rocess 0 received 0
oot process gathered: 0 20 40 60
rocess 1 received 10
rocess 2 received 20
rocess 3 received 30
cem@gcem-OptiPlex-Tower-7010:~$
```

PROGRAM - 9

Write a MPI Program to demonstration of MPI_Reduce and MPI_Allreduce (MPI_MAX, MPI_MIN, MPI_SUM, MPI_PROD)

PROGRAM:

```
#include <stdio.h>
 #include <mpi.h>
int main(int argc, char *argv[])
{
   int rank, size;
   int value, sum, max;

   MPI_Init(&argc, &argv);                    // Initialize MPI
   MPI_Comm_rank(MPI_COMM_WORLD, &rank);      // Get process rank
   MPI_Comm_size(MPI_COMM_WORLD, &size);      // Get total number of processes

   value = rank + 1;  // Example local value: 1, 2, 3, ...

   // ----------------  MPI_Reduce for sum ----------------
   MPI_Reduce(&value, &sum, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);

   if (rank == 0)
   {
      printf("Sum using Reduce: %d \n", sum);
   }

   // ----------------  MPI_Allreduce for max ----------------
   MPI_Allreduce(&value, &max, 1, MPI_INT, MPI_MAX, MPI_COMM_WORLD);

   printf("Max using Allreduce (rank %d): %d \n", rank, max);

   MPI_Finalize();
   return 0;
}
```

SAMPLE OUTPUT: