**Code:**

#include <mpi.h>

#include <stdio.h>

#include <stdlib.h>

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

int rank, size;

int N = 16; // Total number of elements

int array[N];

int local\_sum = 0, total\_sum = 0;

MPI\_Init(&argc, &argv);

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

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

// Check if the number of processes is greater than the number of elements

if (size > N) {

if (rank == 0) {

printf("Error: The number of processes cannot be greater than the number of elements in the array.\n");

printf("Enter the number of processes that is less than or equal to %d.\n", N);

}

MPI\_Finalize();

return 0; // Exit if number of processes is invalid

}

int elements\_per\_proc = N / size;

int remaining\_elements = N % size;

// Initialize the array on rank 0

if (rank == 0) {

for (int i = 0; i < N; i++) {

array[i] = i + 1; // Fill array with values 1 to N

}

printf("Original array: ");

for (int i = 0; i < N; i++) {

printf("%d ", array[i]);

}

printf("\n");

}

int local\_size = elements\_per\_proc + (rank < remaining\_elements ? 1 : 0);

int\* local\_array = (int\*)malloc(local\_size \* sizeof(int));

int displs[size], send\_counts[size];

if (rank == 0) {

int offset = 0;

for (int i = 0; i < size; i++) {

send\_counts[i] = elements\_per\_proc + (i < remaining\_elements ? 1 : 0);

displs[i] = offset;

offset += send\_counts[i];

}

}

// Scatter the array to all processes

MPI\_Scatterv(array, send\_counts, displs, MPI\_INT, local\_array, local\_size, MPI\_INT, 0, MPI\_COMM\_WORLD);

// Each process calculates its local sum

for (int i = 0; i < local\_size; i++) {

local\_sum += local\_array[i];

}

free(local\_array);

// Print the local sum for each process

printf("Processor %d calculated local sum: %d\n", rank, local\_sum);

// Use MPI\_Reduce to get the total sum at root (rank 0)

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

// Rank 0 prints the total sum

if (rank == 0) {

printf("Total sum of array: %d\n", total\_sum);

}

MPI\_Finalize();

return 0;

}

