Practical Assignment 4(Based on Unit 5)

Write an MPI program to calculate sum of randomly generated 1000 numbers (stored in array) on a cluster

Program-

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

#define N 1000 // Total number of random numbers
#define SEED 12345 // Seed for random number generator

int main(int argc, char **argv) {
   int rank, size;
   int i, local_sum = 0, total_sum = 0;
   int local_numbers[N/2]; // Assuming 2 processes for simplicity (you can generalize this)
   int global_numbers[N];

   // Initialize MPI
   MPI_Init(&argc, &argv);
   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 (size != 2) {
      if (rank == 0) {
         printf("This program is intended to run with 2 processes.\n");
      }
      MPI_Finalize();
      return 0;
   }

   srand(SEED + rank); // Set different seed for each process

   // Divide the work: each process will work on N/2 numbers
   for (i = 0; i < N / 2; i++) {
      local_numbers[i] = rand() % 100; // Random number between 0 and 99
      local_sum += local_numbers[i];   // Compute the partial sum
   }

   // Print the local sum of each process
   printf("Process %d, Local sum: %d\n", rank, local_sum);

   // Reduce operation: Sum all local sums into total_sum at root process (rank 0)
   MPI_Reduce(&local_sum, &total_sum, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);
```

```
    // Only process 0 will print the total sum
    if (rank == 0) {
        printf("Total sum of all numbers: %d\n", total_sum);
    }

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

Write an MPI program to calculate sum and average of randomly generated 1000 numbers (stored in array) on a cluster

Program-

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

#define N 1000 // Total number of random numbers
#define SEED 12345 // Seed for random number generator

int main(int argc, char **argv) {
    int rank, size;
    int i, local_sum = 0, total_sum = 0;
    float local_avg, total_avg;
    int numbers_per_process;
    int *local_numbers;

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

    // Calculate the number of numbers each process will handle
    numbers_per_process = N / size;

    // Allocate memory for local numbers
    local_numbers = (int*)malloc(numbers_per_process * sizeof(int));

    // Seed the random number generator differently for each process
    srand(SEED + rank);

    // Each process generates its part of the numbers
    for (i = 0; i < numbers_per_process; i++) {
        local_numbers[i] = rand() % 100; // Random number between 0 and 99
        local_sum += local_numbers[i];    // Compute the partial sum
    }

    // Calculate the local average for this process
    local_avg = (float)local_sum / numbers_per_process;

    // Print the local sum and average for each process (optional for debugging)
    printf("Process %d: Local sum = %d, Local average = %.2f\n", rank, local_sum, local_avg);

    // Reduce operation: Sum all local sums into total_sum at root process (rank 0)
    MPI_Reduce(&local_sum, &total_sum, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);

    // The root process (rank 0) calculates the global average
    if (rank == 0) {
        total_avg = (float)total_sum / N; // Total average
        printf("Total sum = %d\n", total_sum);
        printf("Total average = %.2f\n", total_avg);
    }

    // Clean up
    free(local_numbers);

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

Write an MPI program to find the max number from randomly generated 1000 numbers (stored in array) on a cluster (Hint: Use MPI_Reduce)

Pragram-

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

#define N 1000 // Total number of random numbers
#define SEED 12345 // Seed for random number generator

int main(int argc, char **argv) {
    int rank, size;
    int i, local_max, global_max;
    int numbers_per_process;
    int *local_numbers;
    // Initialize MPI
    MPI_Init(&argc, &argv);
    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

    // Calculate the number of numbers each process will handle
    numbers_per_process = N / size;

    // Allocate memory for local numbers
    local_numbers = (int*)malloc(numbers_per_process * sizeof(int));

    // Seed the random number generator differently for each process
    srand(SEED + rank);

    // Each process generates its part of the numbers
    local_max = -1;  // Initialize local_max to a value lower than any possible random number
    for (i = 0; i < numbers_per_process; i++) {
        local_numbers[i] = rand() % 100; // Random number between 0 and 99
        if (local_numbers[i] > local_max) {
            local_max = local_numbers[i]; // Update the local max
        }
    }

    // Print the local max for debugging (optional)
    printf("Process %d: Local max = %d\n", rank, local_max);

    // Use MPI_Reduce to find the maximum value across all processes
    MPI_Reduce(&local_max, &global_max, 1, MPI_INT, MPI_MAX, 0, MPI_COMM_WORLD);

    // Only the root process (rank 0) will print the global maximum
    if (rank == 0) {
        printf("Global max = %d\n", global_max);
    }
    // Clean up
    free(local_numbers);

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

Write an MPI program to find the min number from randomly generated 1000 numbers (stored in array) on a cluster (Hint: Use MPI_Reduce)

Program-

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

#define N 1000 // Total number of random numbers
#define SEED 12345 // Seed for random number generator

int main(int argc, char **argv) {
    int rank, size;
    int i, local_min, global_min;
    int numbers_per_process;
    int *local_numbers;

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

    // Calculate the number of numbers each process will handle
    numbers_per_process = N / size;

    // Allocate memory for local numbers
    local_numbers = (int*)malloc(numbers_per_process * sizeof(int));
    // Seed the random number generator differently for each process
    srand(SEED + rank);

    // Each process generates its part of the numbers
    local_min = 100;  // Initialize local_min to a value higher than the max random
number
    for (i = 0; i < numbers_per_process; i++) {
        local_numbers[i] = rand() % 100; // Random number between 0 and 99
        if (local_numbers[i] < local_min) {
            local_min = local_numbers[i]; // Update the local min
        }
    }

    // Print the local min for debugging (optional)
    printf("Process %d: Local min = %d\n", rank, local_min);

    // Use MPI_Reduce to find the minimum value across all processes
    MPI_Reduce(&local_min, &global_min, 1, MPI_INT, MPI_MIN, 0,
MPI_COMM_WORLD);

    // Only the root process (rank 0) will print the global minimum
    if (rank == 0) {
        printf("Global min = %d\n", global_min);
    }

    // Clean up
    free(local_numbers);

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

Write an MPI program to calculate sum of all even randomly generated 1000 numbers (stored in array) on a cluster

Program-

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

```
#include <stdlib.h>
#include <mpi.h>
#include <time.h>

#define N 1000 // Total number of random numbers
#define SEED 12345 // Seed for random number generator

int main(int argc, char **argv) {
   int rank, size;
   int i, local_sum = 0, global_sum = 0;
   int numbers_per_process;
   int *local_numbers;

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

   // Calculate the number of numbers each process will handle
   numbers_per_process = N / size;

   // Allocate memory for local numbers
   local_numbers = (int*)malloc(numbers_per_process * sizeof(int));

   // Seed the random number generator differently for each process
   srand(SEED + rank);

   // Each process generates its part of the numbers and computes the local sum
of even numbers
   for (i = 0; i < numbers_per_process; i++) {
      local_numbers[i] = rand() % 100; // Random number between 0 and 99
      if (local_numbers[i] % 2 == 0) { // Check if the number is even
         local_sum += local_numbers[i]; // Add the even number to the local sum
      }
   }
```

```
    // Print the local sum for debugging (optional)
    printf("Process %d: Local sum of even numbers = %d\n", rank, local_sum);

    // Use MPI_Reduce to sum all local sums of even numbers into global_sum at
root process (rank 0)
    MPI_Reduce(&local_sum, &global_sum, 1, MPI_INT, MPI_SUM, 0,
MPI_COMM_WORLD);

    // Only the root process (rank 0) will print the global sum of even numbers
    if (rank == 0) {
        printf("Total sum of even numbers = %d\n", global_sum);
    }

    // Clean up
    free(local_numbers);

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

Write an MPI program to calculate sum of all odd randomly generated 1000 numbers (stored in array) on a cluster.

Program-

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

#define N 1000 // Total number of random numbers
#define SEED 12345 // Seed for random number generator

int main(int argc, char **argv) {
    int rank, size;
    int i, local_sum = 0, global_sum = 0;
    int numbers_per_process;
    int *local_numbers;
    // Initialize MPI
    MPI_Init(&argc, &argv);
    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

    // Calculate the number of numbers each process will handle
    numbers_per_process = N / size;

    // Allocate memory for local numbers
    local_numbers = (int*)malloc(numbers_per_process * sizeof(int));

    // Seed the random number generator differently for each process
    srand(SEED + rank);

    // Each process generates its part of the numbers and computes the local sum of odd numbers
    for (i = 0; i < numbers_per_process; i++) {
        local_numbers[i] = rand() % 100; // Random number between 0 and 99
        if (local_numbers[i] % 2 != 0) { // Check if the number is odd
            local_sum += local_numbers[i]; // Add the odd number to the local sum
        }
    }

    // Print the local sum for debugging (optional)
    printf("Process %d: Local sum of odd numbers = %d\n", rank, local_sum);

    // Use MPI_Reduce to sum all local sums of odd numbers into global_sum at root process (rank 0)
    MPI_Reduce(&local_sum, &global_sum, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);

    // Only the root process (rank 0) will print the global sum of odd numbers
    if (rank == 0) {
        printf("Total sum of odd numbers = %d\n", global_sum);
    }
    // Clean up
    free(local_numbers);

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