1.WAP in MPI distributed summation of the numbers from 1 to 100 using 4 processes

```
#include <stdio.h>
#include <mpi.h>
int main(int argc, char **argv) {
int rank, size;
int x[100];
int s = 0, a, b, c;
int tag = 100;
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);

if (rank == 0) {
for (int i = 0; i < 100; i++) {
x[i] = i + 1;
}
for (int i = 0; i < 25; i++) {
s += x[i];
}

printf("Sum of numbers in rank 0 = %d\n", s);
MPI_Send(&x[25], 25, MPI_INT, 1, tag, MPI_COMM_WORLD);
MPI_Send(&x[50], 25, MPI_INT, 2, tag, MPI_COMM_WORLD);
MPI_Send(&x[75], 25, MPI_INT, 3, tag, MPI_COMM_WORLD);

MPI_Recv(&a, 1, MPI_INT, 1, tag, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
MPI_Recv(&b, 1, MPI_INT, 2, tag, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
MPI_Recv(&c, 1, MPI_INT, 3, tag, MPI_COMM_WORLD, MPI_STATUS_IGNORE);

int total_sum = s + a + b + c;
printf("\nFinal sum = %d\n", total_sum);
}

if (rank == 1 || rank == 2 || rank == 3) {
int y[25];
MPI_Recv(&y, 25, MPI_INT, 0, tag, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
```

```
printf("\nValues in rank %d:\n", rank);
for (int i = 0; i < 25; i++) {
printf("%d ", y[i]);
s += y[i];
}
printf("\nSum of numbers in rank %d = %d\n", rank, s);
MPI_Send(&s, 1, MPI_INT, 0, tag, MPI_COMM_WORLD);
}

MPI_Finalize();
return 0;
}
```

OUTPUT:

Sum of numbers in rank 0 = 325

Final sum = 5050

Values in rank 1:
26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
Sum of numbers in rank 1 = 950

Values in rank 2:
51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75
Sum of numbers in rank 2 = 1575

Values in rank 3:
76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100
Sum of numbers in rank 3 = 2200

2.WAP in MPI sum of numbers from 1 to 20 using 4 processes and MPI_Bcast + MPI_Send/MPI_Recv

```
#include <stdio.h>
#include <mpi.h>
int main(int argc, char **argv) {
int rank, x[20], local_sum = 0;
int tag = 123, total_sum, temp;
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
if (rank == 0) {
for (int i = 0; i < 20; i++) {
x[i] = i + 1;
```

```
printf("%d ", x[i]);
}
printf("\n");
}
MPI_Bcast(x, 20, MPI_INT, 0, MPI_COMM_WORLD);
int start = rank * 5;
int end = start + 5;
for (int i = start; i < end; i++) {
local_sum += x[i];
}
printf("Rank %d local sum = %d\n", rank, local_sum);
if (rank != 0) {
MPI_Send(&local_sum, 1, MPI_INT, 0, tag, MPI_COMM_WORLD);
}
if (rank == 0) {
total_sum = local_sum;
for (int src = 1; src <= 3; src++) {
MPI_Recv(&temp, 1, MPI_INT, src, tag, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
total_sum += temp;
}
printf("\nFinal sum = %d\n", total_sum);
}
MPI_Finalize();
return 0;
}
```

OUTPUT:

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
Rank 0 local sum = 15

Final sum = 210
Rank 1 local sum = 40
Rank 2 local sum = 65
Rank 3 local sum = 90

3.WAP IN MPI that prints the name of the processor and the rank of each process. The program should identify process 0 as the "boss" and all other processes as "slaves".

```
#include<stdio.h>
#include<mpi.h>
int main(int argc,char** argv)
{
MPI_Init(&argc,&argv);
int rank,procs,len;
char proc_name[1000];
MPI_Comm_rank(MPI_COMM_WORLD,&rank);
MPI_Comm_size(MPI_COMM_WORLD,&procs);
MPI_Get_processor_name(&proc_name,&len);
printf("processor name is %s",proc_name);
printf("\n my rank is %d out of %d\n",rank,procs);
if(rank==0)
{
printf("\ni am the boss\n");
}
else
{
printf("\ni am the slave\n");
}
MPI_Finalize();
}
```

OUTPUT:

processor name is 858abe5dfc3c
my rank is 0 out of 4
i am the boss

processor name is 858abe5dfc3c
my rank is 1 out of 4
i am the slave

processor name is 858abe5dfc3c
my rank is 2 out of 4
i am the slave

processor name is 858abe5dfc3c
my rank is 3 out of 4
i am the slave

4.WAP IN MPI point-to-point communication using MPI_Send and MPI_Recv

```
#include <stdio.h>
#include <mpi.h>
int main(int argc, char **argv) {
int rank, size;
int tag = 100;
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_size(MPI_COMM_WORLD, &size);
if (rank == 0) {
int x[3] = {2, 1, 3};
for (int i = 1; i < size; i++) {
MPI_Send(x, 3, MPI_INT, i, tag, MPI_COMM_WORLD);
printf("Process 0 sent data to process %d\n", i);
}
} else {
int y[3];
MPI_Recv(y, 3, MPI_INT, 0, tag, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
printf("Process %d received data from process 0: ", rank);
for (int i = 0; i < 3; i++) {
printf("%d ", y[i]);
}
printf("\n");
}
MPI_Finalize();
return 0;
}
```

OUTPUT:

Process 0 sent data to process 1
Process 0 sent data to process 2
Process 0 sent data to process 3
Process 1 received data from process 0: 2 1 3
Process 2 received data from process 0: 2 1 3
Process 3 received data from process 0: 2 1 3

5.WAP IN MPI using scatter and gather to find the sum, min, max of:

a) All prime numbers

b) All even numbers

c) All odd numbers

```
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>
#define SIZE 16
int is_prime(int num) {
if (num <= 1) return 0;
for (int i = 2; i * i <= num; i++)
if (num % i == 0) return 0;
return 1;
}
void compute_stats(int *arr, int count, int *min, int *max, int *sum) {
if (count == 0) {
*min = *max = *sum = 0;
return;
}
*min = *max = arr[0];
*sum = 0;
for (int i = 0; i < count; i++) {
if (arr[i] < *min) *min = arr[i];
if (arr[i] > *max) *max = arr[i];
*sum += arr[i];
}
}
int main(int argc, char *argv[]) {
int rank, size;
int data[SIZE] = {11, 18, 3, 7, 9, 14, 23, 21, 31, 5, 19, 6, 10, 17, 13, 4};
int local_data[SIZE / 4];
int local_primes[SIZE / 4], local_evens[SIZE / 4], local_odds[SIZE / 4];
int prime_count = 0, even_count = 0, odd_count = 0;

int prime_counts[4], even_counts[4], odd_counts[4];
int all_primes[SIZE], all_evens[SIZE], all_odds[SIZE];
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
```

```
MPI_Comm_size(MPI_COMM_WORLD, &size);
if (size != 4) {
if (rank == 0)
printf("This program requires exactly 4 processes.\n");
MPI_Finalize();
return 1;
}
MPI_Scatter(data, SIZE / 4, MPI_INT, local_data, SIZE / 4, MPI_INT, 0, MPI_COMM_WORLD);
for (int i = 0; i < SIZE / 4; i++) {
int val = local_data[i];
if (is_prime(val)) local_primes[prime_count++] = val;
if (val % 2 == 0) local_evens[even_count++] = val;
else local_odds[odd_count++] = val;
}
MPI_Gather(&prime_count, 1, MPI_INT, prime_counts, 1, MPI_INT, 0, MPI_COMM_WORLD);
MPI_Gather(&even_count, 1, MPI_INT, even_counts, 1, MPI_INT, 0, MPI_COMM_WORLD);
MPI_Gather(&odd_count, 1, MPI_INT, odd_counts, 1, MPI_INT, 0, MPI_COMM_WORLD);
MPI_Gather(local_primes, SIZE / 4, MPI_INT, all_primes, SIZE / 4, MPI_INT, 0, MPI_COMM_WORLD);
MPI_Gather(local_evens, SIZE / 4, MPI_INT, all_evens, SIZE / 4, MPI_INT, 0, MPI_COMM_WORLD);
MPI_Gather(local_odds, SIZE / 4, MPI_INT, all_odds, SIZE / 4, MPI_INT, 0, MPI_COMM_WORLD);
if (rank == 0) {
int primes[SIZE], evens[SIZE], odds[SIZE];
int p_index = 0, e_index = 0, o_index = 0;
for (int i = 0; i < 4; i++) {
for (int j = 0; j < prime_counts[i]; j++)
primes[p_index++] = all_primes[i * (SIZE / 4) + j];
for (int j = 0; j < even_counts[i]; j++)
evens[e_index++] = all_evens[i * (SIZE / 4) + j];
for (int j = 0; j < odd_counts[i]; j++)
odds[o_index++] = all_odds[i * (SIZE / 4) + j];
}
int p_min, p_max, p_sum;
int e_min, e_max, e_sum;
int o_min, o_max, o_sum;
compute_stats(primes, p_index, &p_min, &p_max, &p_sum);
compute_stats(evens, e_index, &e_min, &e_max, &e_sum);
compute_stats(odds, o_index, &o_min, &o_max, &o_sum);
printf("Primes (%d): ", p_index);
for (int i = 0; i < p_index; i++) printf("%d ", primes[i]);
printf("\nSum = %d, Min = %d, Max = %d\n\n", p_sum, p_min, p_max);
```

```
printf("Evens (%d): ", e_index);
for (int i = 0; i < e_index; i++) printf("%d ", evens[i]);
printf("\nSum = %d, Min = %d, Max = %d\n\n", e_sum, e_min, e_max);
printf("Odds (%d): ", o_index);
for (int i = 0; i < o_index; i++) printf("%d ", odds[i]);
printf("\nSum = %d, Min = %d, Max = %d\n", o_sum, o_min, o_max);
}
MPI_Finalize();
return 0;
}
```

OUTPUT:

Primes (9): 11 3 7 23 31 5 19 17 13
Sum = 129, Min = 3, Max = 31

Evens (5): 18 14 6 10 4
Sum = 52, Min = 4, Max = 18

Odds (11): 11 3 7 9 23 21 31 5 19 17 13
Sum = 159, Min = 3, Max = 31

6.WAP IN MPI To perform matrix multiplication

```
#include <mpi.h>
#include <stdio.h>
int main(int argc, char** argv) {
int rank;
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
int arow = 3, acol = 3,brow = 3, bcol = 3;
int A[arow][acol], B[brow][bcol], C[arow][bcol];
if (rank == 0) {
int v=1;
for (int i = 0; i < arow; i++)
for (int j = 0; j < acol; j++)
A[i][j] = v++;
MPI_Send(A,9, MPI_INT, 2, 123, MPI_COMM_WORLD);
} else if (rank == 1) {
int v=10;
for (int i = 0; i < brow; i++)
```

```
for (int j = 0; j < bcol; j++)
B[i][j] = v++;
MPI_Send(B, brow * bcol, MPI_INT, 2, 123, MPI_COMM_WORLD);
} else if (rank == 2) {
MPI_Recv(A,9, MPI_INT, 0, 123, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
MPI_Recv(B,9, MPI_INT, 1, 123, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
for (int i = 0; i < arow; i++) {
for (int j = 0; j < bcol; j++) {
C[i][j] = 0;
for (int k = 0; k < acol; k++) {
C[i][j] += A[i][k] * B[k][j];
}
}
}
printf("Matrix C:\n");
for (int i = 0; i < arow; i++) {
for (int j = 0; j < bcol; j++) {
printf("%d ", C[i][j]);
}
printf("\n");
}
}
MPI_Finalize();
return 0;
}
```

OUTPUT:

```
Matrix C:
84 90 96
201 216 231
318 342 366
```

7.WAP IN MPI to perform linear search

```
#include <stdio.h>
#include <mpi.h>
int main(int argc, char *argv[]) {
int rank, size;
int data[8] = {5, 10, 15, 20, 25, 30, 35, 40};
int key = 25;
int found = 0;
```

```
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_size(MPI_COMM_WORLD, &size);
int chunk = 8 / size;
int subdata[chunk];
MPI_Scatter(data, chunk, MPI_INT, subdata, chunk, MPI_INT, 0, MPI_COMM_WORLD);
MPI_Bcast(&key, 1, MPI_INT, 0, MPI_COMM_WORLD);
int local_found = 0;
for (int i = 0; i < chunk; i++) {
if (subdata[i] == key) {
int index = rank * chunk + i;
printf("Key %d found at index %d by process %d\n", key, index, rank);
local_found = 1;
}
}
MPI_Reduce(&local_found, &found, 1, MPI_INT, MPI_MAX, 0, MPI_COMM_WORLD);
if (rank == 0 && found == 0)
printf("Key %d was not found in any process.\n", key);
MPI_Finalize();
return 0;
}
```

Output:

Execution ...

Key 25 found at index 4 by process 2

8. WAP IN MPI to add 3 matrix

```
#include <stdio.h>
#include <mpi.h>
int main(int argc, char *argv[]) {
int rank, size;
int A[2][2] = {{1, 2}, {3, 4}};
int B[2][2] = {{5, 6}, {7, 8}};
int C[2][2] = {{9, 10}, {11, 12}};
int local_sum[2][2];
```

```
int final_sum[2][2];
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_size(MPI_COMM_WORLD, &size);
for (int i = 0; i < 2; i++) {
for (int j = 0; j < 2; j++) {
local_sum[i][j] = A[i][j] + B[i][j] + C[i][j];
}
}
MPI_Reduce(local_sum, final_sum, 4, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);
if (rank == 0) {
printf("Sum of 3 matrices:\n");
for (int i = 0; i < 2; i++) {
for (int j = 0; j < 2; j++) {
printf("%4d ", final_sum[i][j]);
}
printf("\n");
}
}

MPI_Finalize();
return 0;
}
```

OUTPUT:

Sum of 3 matrices:

60 72

84 96

9.WAP IN MPI to sum array using reduce

```
#include <stdio.h>
#include <mpi.h>
int main(int argc, char** argv) {
int rank, size;
int local[5], global[5];
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_size(MPI_COMM_WORLD, &size);
```

```
if (size != 4) {
if (rank == 0)
printf("Please run the program with 4 processes.\n");
MPI_Finalize();
return 0;
}
for (int i = 0; i < 5; i++) {
local[i] = (rank + 1) * (i + 1);
}
printf("Process %d local array: ", rank);
for (int i = 0; i < 5; i++) {
printf("%d ", local[i]);
}
printf("\n");
MPI_Reduce(local, global, 5, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);
if (rank == 0) {
printf("\nFinal summed array at process 0: ");
for (int i = 0; i < 5; i++) {
printf("%d ", global[i]);
}
printf("\n");
}
MPI_Finalize();
return 0;
}
```

OUTPUT:

Process 0 local array: 1 2 3 4 5

Final summed array at process 0: 10 20 30 40 50
Process 1 local array: 2 4 6 8 10
Process 2 local array: 3 6 9 12 15
Process 3 local array: 4 8 12 16 20