**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