**Name - Omkar\_Ohol  
Roll\_No - 2213425  
Sub - Hpc\_Lab  
  
  
1)**

**Code -**

#include <mpi.h>

#include <stdio.h>

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

int rank, size, sum = 0, global\_sum;

// Initialize MPI environment

MPI\_Init(&argc, &argv);

// Get the rank and size

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

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

// Each process contributes its rank to the sum

sum = rank;

// Compute the global sum (one process will get the result)

MPI\_Reduce(&sum, &global\_sum, 1, MPI\_INT, MPI\_SUM, 0, MPI\_COMM\_WORLD);

if (rank == 0) {

printf("Sum of ranks (process 0): %d\n", global\_sum);

}

// Broadcast the result to all processes

MPI\_Bcast(&global\_sum, 1, MPI\_INT, 0, MPI\_COMM\_WORLD);

printf("Process %d received the sum: %d\n", rank, global\_sum);

// Finalize MPI environment

MPI\_Finalize();

return 0;

}

**Output::**

Sum of ranks (process 0): 6

Process 0 received the sum: 6

Process 1 received the sum: 6

Process 2 received the sum: 6

Process 3 received the sum: 6

**2)** C program to multiply two matrices and store the result in a third matrix**.**

**Code -**#include <stdio.h>

#define ROW\_A 2 // Number of rows in Matrix A

#define COL\_A 2 // Number of columns in Matrix A

#define COL\_B 2 // Number of columns in Matrix B

int main() {

int A[ROW\_A][COL\_A] = {{1, 2}, {3, 4}}; // Initialize Matrix A

int B[COL\_A][COL\_B] = {{5, 6}, {7, 8}}; // Initialize Matrix B

int C[ROW\_A][COL\_B] = {0}; // Initialize result matrix C with zeros

// Matrix multiplication logic

for (int i = 0; i < ROW\_A; i++) {

for (int j = 0; j < COL\_B; j++) {

for (int k = 0; k < COL\_A; k++) {

C[i][j] += A[i][k] \* B[k][j];

}

}

}

// Print Matrix A

printf("Matrix A:\n");

for (int i = 0; i < ROW\_A; i++) {

for (int j = 0; j < COL\_A; j++) {

printf("%d ", A[i][j]);

}

printf("\n");

}

// Print Matrix B

printf("\nMatrix B:\n");

for (int i = 0; i < COL\_A; i++) {

for (int j = 0; j < COL\_B; j++) {

printf("%d ", B[i][j]);

}

printf("\n");

}

// Print Matrix C

printf("\nMatrix C (Result):\n");

for (int i = 0; i < ROW\_A; i++) {

for (int j = 0; j < COL\_B; j++) {

printf("%d ", C[i][j]);

}

printf("\n");

}

return 0;

}

**OUTPUT -  
  
1 2**

**3 4**

**5 6**

**7 8**

**19 22**

**43 50**

**3)** MPI program to calculate the value of π using the Dartboard Algorithm **Code -**#include <mpi.h>

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

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

int rank, size, i;

long long int num\_points = 1000000; // Total number of points (adjustable)

long long int points\_in\_circle = 0, total\_points\_in\_circle = 0;

// Initialize MPI

MPI\_Init(&argc, &argv);

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

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

// Seed the random number generator differently for each process

srand(time(NULL) + rank);

// Number of points each process will calculate

long long int points\_per\_process = num\_points / size;

// Monte Carlo simulation

for (i = 0; i < points\_per\_process; i++) {

double x = (double)rand() / RAND\_MAX; // Random x in [0, 1]

double y = (double)rand() / RAND\_MAX; // Random y in [0, 1]

if (x \* x + y \* y <= 1.0) { // Check if the point lies inside the circle

points\_in\_circle++;

}

}

// Reduce all points\_in\_circle to the root process

MPI\_Reduce(&points\_in\_circle, &total\_points\_in\_circle, 1, MPI\_LONG\_LONG, MPI\_SUM, 0, MPI\_COMM\_WORLD);

// Calculate and print π in the root process

if (rank == 0) {

double pi = 4.0 \* (double)total\_points\_in\_circle / num\_points;

printf("Estimated value of Pi: %f\n", pi);

}

// Finalize MPI

MPI\_Finalize();

return 0;

}

**OUTPUT -**

Estimated value of Pi: 3.141592

**4).** **C program** to calculate the value of π using the **Dartboard Algorithm** without parallel processing.  
  
**Code -**#include <stdio.h>

#include <stdlib.h>

#include <time.h>

int main() {

long long int num\_points = 1000000; // Total number of random points

long long int points\_in\_circle = 0;

double x, y;

// Seed the random number generator

srand(time(NULL));

// Monte Carlo simulation

for (long long int i = 0; i < num\_points; i++) {

x = (double)rand() / RAND\_MAX; // Random x in [0, 1]

y = (double)rand() / RAND\_MAX; // Random y in [0, 1]

// Check if the point lies inside the circle

if (x \* x + y \* y <= 1.0) {

points\_in\_circle++;

}

}

// Calculate π using the formula: π = 4 \* (points inside circle / total points)

double pi = 4.0 \* (double)points\_in\_circle / num\_points;

// Print the result

printf("Estimated value of Pi: %f\n", pi);

return 0;

}

**Output -**

Estimated value of Pi: 3.141872

**5)** MPI program to determine the partner process and exchange messages (your name or a number) between two processes using blocking send and receive routines**.  
  
Code -**

#include <mpi.h>

#include <stdio.h>

#include <string.h>

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

int rank, size;

char message[100];

MPI\_Status status;

// Initialize MPI

MPI\_Init(&argc, &argv);

// Get the rank (process ID) and size (total number of processes)

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

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

if (size < 2) {

printf("Please run this program with at least 2 processes.\n");

MPI\_Finalize();

return 0;

}

if (rank == 0) { // Process 0

// Create a message and send it to process 1

strcpy(message, "Hello from Process 0");

MPI\_Send(message, strlen(message) + 1, MPI\_CHAR, 1, 0, MPI\_COMM\_WORLD);

// Receive a message from process 1

MPI\_Recv(message, 100, MPI\_CHAR, 1, 0, MPI\_COMM\_WORLD, &status);

printf("Process 0 received: %s\n", message);

} else if (rank == 1) { // Process 1

// Receive a message from process 0

MPI\_Recv(message, 100, MPI\_CHAR, 0, 0, MPI\_COMM\_WORLD, &status);

printf("Process 1 received: %s\n", message);

// Create a message and send it to process 0

strcpy(message, "Hello from Process 1");

MPI\_Send(message, strlen(message) + 1, MPI\_CHAR, 0, 0, MPI\_COMM\_WORLD);

}

// Finalize MPI

MPI\_Finalize();

return 0;

}

**Output -**  
  
Process 1 received: Hello from Process 0

Process 0 received: Hello from Process 1

**6)**  
**MPI program** to check if the number of processes is even. If the number of processes is even, the program prints your name from each process. Otherwise, it prints an error message and exits.  
 **Code-**

#include <mpi.h>

#include <stdio.h>

#include <stdlib.h>

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

int rank, size;

// Initialize MPI

MPI\_Init(&argc, &argv);

// Get the rank (process ID) and size (total number of processes)

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

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

// Check if the number of processes is even

if (size % 2 != 0) {

if (rank == 0) { // Only the root process prints the error message

printf("Error: The number of processes is odd. Please use an even number of processes.\n");

}

MPI\_Finalize();

return 1; // Exit with error

}

// Print your name from each process

printf("Process %d: Hello, my name is Omkar!\n", rank);

// Finalize MPI

MPI\_Finalize();

return 0;

}

**Output -**

Process 0: Hello, my name is Omkar!

Process 1: Hello, my name is Omkar!

Process 2: Hello, my name is Omkar!

Process 3: Hello, my name is Omkar!

Error: The number of processes is odd. Please use an even number of processes.