# Final Year B. Tech., Sem VII 2024-25

High Performance Computing Lab

**Practical No. 9**

## Implement Matrix-Vector Multiplication using MPI. Use different number of processes and analyze the performance.

CODE :

#include <mpi.h>

#include <stdio.h>

#include <stdlib.h>

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

MPI\_Init(&argc, &argv);

int world\_size, world\_rank;

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

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

int m = 5;

int n = 5;

int \*A = NULL;

int \*x = (int\*)malloc(n \* sizeof(int));

int \*local\_A;

int local\_m = m / world\_size;

if (world\_rank == 0) {

A = (int\*)malloc(m \* n \* sizeof(int));

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

for (int j = 0; j < n; j++) {

A[i \* n + j] = i + j;

}

}

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

x[i] = i + 1;

}

}

local\_A = (int\*)malloc(local\_m \* n \* sizeof(int));

double start\_time = MPI\_Wtime();

MPI\_Scatter(A, local\_m \* n, MPI\_INT, local\_A, local\_m \* n, MPI\_INT, 0, MPI\_COMM\_WORLD);

MPI\_Bcast(x, n, MPI\_INT, 0, MPI\_COMM\_WORLD);

int\* local\_y = (int\*)malloc(local\_m \* sizeof(int));

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

local\_y[i] = 0;

for (int j = 0; j < n; j++) {

local\_y[i] += local\_A[i \* n + j] \* x[j];

}

}

int\* y = NULL;

if (world\_rank == 0) {

y = (int\*)malloc(m \* sizeof(int));

}

MPI\_Gather(local\_y, local\_m, MPI\_INT, y, local\_m, MPI\_INT, 0, MPI\_COMM\_WORLD);

double end\_time = MPI\_Wtime();

if (world\_rank == 0) {

printf("Resultant vector y = A \* x:\n");

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

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

}

printf("\n");

printf("Total execution time with %d processes: %f seconds\n", world\_size, end\_time - start\_time);

}

free(local\_A);

free(local\_y);

free(x);

if (world\_rank == 0) {

free(A);

free(y);

}

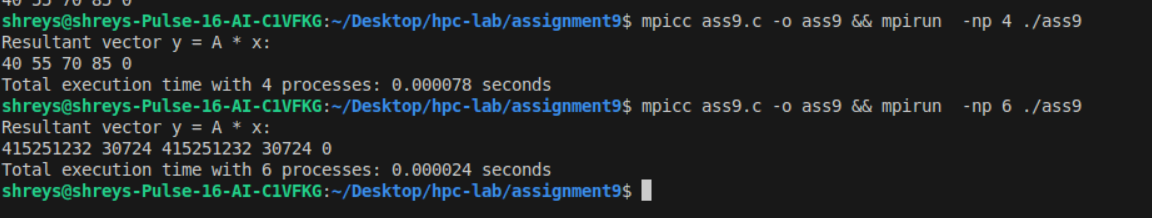
MPI\_Finalize();

return 0;

}

Passing processes through cmd

for 4 and 6 processes clear diffrence is there in execution time



## Implement Matrix-Matrix Multiplication using MPI. Use different number of processes and analyze the performance.

CODE :

#include <mpi.h>

#include <stdio.h>

#include <stdlib.h>

#define N 4

void matrixMultiply(int\* A\_local, int\* B, int\* C\_local, int rows\_per\_proc, int n) {

for (int i = 0; i < rows\_per\_proc; i++) {

for (int j = 0; j < n; j++) {

C\_local[i \* n + j] = 0;

for (int k = 0; k < n; k++) {

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

}

}

}

}

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

int rank, size;

MPI\_Init(&argc, &argv);

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

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

int rows\_per\_proc = N / size;

int\* A = NULL;

int\* B = (int\*)malloc(N \* N \* sizeof(int));

int\* C = NULL;

if (rank == 0) {

A = (int\*)malloc(N \* N \* sizeof(int));

C = (int\*)malloc(N \* N \* sizeof(int));

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

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

A[i \* N + j] = i + j;

B[i \* N + j] = i \* j;

}

}

}

int\* A\_local = (int\*)malloc(rows\_per\_proc \* N \* sizeof(int));

int\* C\_local = (int\*)malloc(rows\_per\_proc \* N \* sizeof(int));

double start\_time = MPI\_Wtime();

MPI\_Scatter(A, rows\_per\_proc \* N, MPI\_INT, A\_local, rows\_per\_proc \* N, MPI\_INT, 0, MPI\_COMM\_WORLD);

MPI\_Bcast(B, N \* N, MPI\_INT, 0, MPI\_COMM\_WORLD);

matrixMultiply(A\_local, B, C\_local, rows\_per\_proc, N);

MPI\_Gather(C\_local, rows\_per\_proc \* N, MPI\_INT, C, rows\_per\_proc \* N, MPI\_INT, 0, MPI\_COMM\_WORLD);

double end\_time = MPI\_Wtime();

if (rank == 0) {

printf("Resulting Matrix C:\n");

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

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

printf("%d ", C[i \* N + j]);

}

printf("\n");

}

printf("Total execution time with %d processes: %f seconds\n", size, end\_time - start\_time);

}

free(A\_local);

free(C\_local);

free(B);

if (rank == 0) {

free(A);

free(C);

}

MPI\_Finalize();

return 0;

}

Execution with 4 as no. of processes and 4 respectively

