# Final Year B. Tech., Sem VII 2025-26

High Performance Computing Lab

**Practical No. 7**

**Name: Ganesh Chavhan**

**PRN: 22510023**

**Batch: B4**

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

Code:

*// Matrix-Vector Multiplication*

#include <mpi.h>

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#define N 4

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

*int* rank**,** size**;**

*int* A[N][N]**,** x[N]**,** y[N]**;**

    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\_sub[rows\_per\_proc][N]**;**

*int* y\_sub[rows\_per\_proc]**;**

*// Initialize matrix and vector only in rank 0*

**if** (rank **==** 0) {

        srand(time(NULL))**;**

        printf("Matrix A:\n")**;**

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

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

                A[i][j] **=** rand() **%** 10**;**

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

            }

            printf("\n")**;**

        }

        printf("Vector x:\n")**;**

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

            x[i] **=** rand() **%** 10**;**

            printf("*%d* "**,** x[i])**;**

        }

        printf("\n")**;**

    }

*// Scatter rows of A to all processes*

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

*// Broadcast vector x to all processes*

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

*// Local computation*

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

        y\_sub[i] **=** 0**;**

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

            y\_sub[i] **+=** a\_sub[i][j] **\*** x[j]**;**

    }

*// Gather results*

    MPI\_Gather(y\_sub**,** rows\_per\_proc**,** MPI\_INT**,** y**,** rows\_per\_proc**,** MPI\_INT**,** 0**,** MPI\_COMM\_WORLD)**;**

**if** (rank **==** 0) {

        printf("Result Vector y = A \* x:\n")**;**

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

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

        printf("\n")**;**

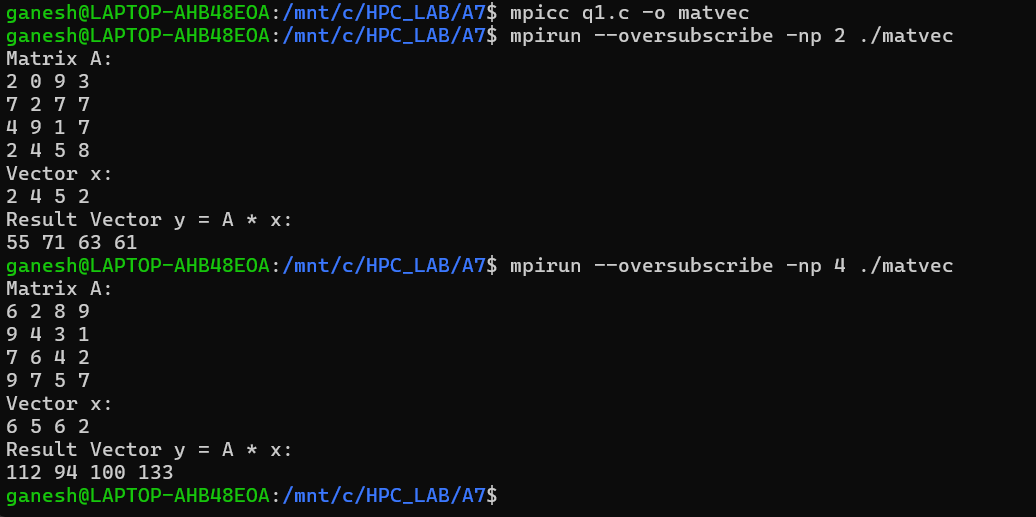
    }

    MPI\_Finalize()**;**

**return** 0**;**

}

Output:



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

Code:

*//  matrix matrix multiplication*

#include <mpi.h>

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#define N 4

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

*int* rank**,** size**;**

*int* A[N][N]**,** B[N][N]**,** C[N][N]**;**

    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\_sub[rows\_per\_proc][N]**;**

*int* c\_sub[rows\_per\_proc][N]**;**

*// Initialize matrices in rank 0*

**if** (rank **==** 0) {

        srand(time(NULL))**;**

        printf("Matrix A:\n")**;**

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

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

                A[i][j] **=** rand() **%** 10**;**

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

**if** (j **==** N**-**1) printf("\n")**;**

            }

        printf("Matrix B:\n")**;**

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

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

                B[i][j] **=** rand() **%** 10**;**

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

**if** (j **==** N**-**1) printf("\n")**;**

            }

    }

*// Scatter rows of A*

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

*// Broadcast matrix B*

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

*// Local computation*

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

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

            c\_sub[i][j] **=** 0**;**

**for** (*int* k **=** 0**;** k **<** N**;** k**++**)

                c\_sub[i][j] **+=** a\_sub[i][k] **\*** B[k][j]**;**

        }

*// Gather results*

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

**if** (rank **==** 0) {

        printf("Result Matrix C = A \* B:\n")**;**

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

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

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

            printf("\n")**;**

        }

    }

    MPI\_Finalize()**;**

**return** 0**;**

}

Output:

