Final Year B. Tech., Sem VII 2024

High_Performance_Computing_Lab

Practical No. 9

Abhijeet Kamalekar

2020BTECS00010

---

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

```
#include <mpi.h>

#include <stdio.h>
#include <stdlib.h>

void matrix_vector_multiply(double* matrix, double* vector, double* result, int rows, int
cols) {
for (int i = 0; i < rows; i++) {
result[i] = 0;
for (int j = 0; j < cols; j++) {
result[i] += matrix[i * cols + j] * vector[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 = 4; // Total number of rows in the matrix
int cols = 4; // Total number of columns in the matrix

double matrix[rows * cols];
double vector[cols];
double local_result[rows / size];
double final_result[rows];

if (rank == 0) {
// Initialize the matrix and vector
for (int i = 0; i < rows; i++) {
```

```c
for (int j = 0; j < cols; j++) {
matrix[i * cols + j] = i + j; // Sample values
}
}
for (int i = 0; i < cols; i++) {
vector[i] = 1; // Sample vector
}
}

// Broadcast the vector to all processes
MPI_Bcast(vector, cols, MPI_DOUBLE, 0, MPI_COMM_WORLD);

// Scatter the matrix rows to all processes
MPI_Scatter(matrix, rows / size * cols, MPI_DOUBLE,
local_result, rows / size * cols, MPI_DOUBLE,
0, MPI_COMM_WORLD);

// Each process computes its local result
matrix_vector_multiply(local_result, vector, local_result, rows / size, cols);

// Gather results from all processes
MPI_Gather(local_result, rows / size, MPI_DOUBLE,
final_result, rows / size, MPI_DOUBLE,
0, MPI_COMM_WORLD);

if (rank == 0) {
// Print the result
printf("Result vector:\n");
for (int i = 0; i < rows; i++) {
printf("%f\n", final_result[i]);
}
}

MPI_Finalize();
return 0;
}
```

output:

```
abhijeet@abhijeet-VirtualBox:~/MPI$ mpirun -np 4 ./9_1
Result vector:
8.000000
15.000000
24.000000
35.000000
```

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

```
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>

void multiply_matrices(double* A, double* B, double* C, int rows_A, int cols_A, int
cols_B) {
for (int i = 0; i < rows_A; i++) {
for (int j = 0; j < cols_B; j++) {
C[i * cols_B + j] = 0;
for (int k = 0; k < cols_A; k++) {
C[i * cols_B + j] += A[i * cols_A + k] * B[k * cols_B + j];
}
}
}
}

int main(int argc, char** argv) {
MPI_Init(&argc, &argv);
int rank, size;
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_size(MPI_COMM_WORLD, &size);

const int N = 4; // Size of the matrices (N x N)
double A[N][N], B[N][N], C[N][N], local_C[N / size][N];

if (rank == 0) {
// Initialize matrices A and B
for (int i = 0; i < N; i++) {
for (int j = 0; j < N; j++) {
A[i][j] = 1.0; // Fill with sample values
B[i][j] = 1.0; // Fill with sample values
}
}
}

// Broadcast matrix B to all processes
MPI_Bcast(B, N * N, MPI_DOUBLE, 0, MPI_COMM_WORLD);
// Scatter rows of matrix A to all processes
MPI_Scatter(A, N / size * N, MPI_DOUBLE, local_C, N / size * N, MPI_DOUBLE, 0,
```

```
MPI_COMM_WORLD);

// Each process computes its portion of the result matrix
multiply_matrices(local_C, B, local_C, N / size, N, N);

// Gather the results back to the root process
MPI_Gather(local_C, N / size * N, MPI_DOUBLE, C, N / size * N, MPI_DOUBLE, 0,
MPI_COMM_WORLD);

if (rank == 0) {
// Print the resulting matrix C
printf("Resulting Matrix C:\n");
for (int i = 0; i < N; i++) {
for (int j = 0; j < N; j++) {
printf("%f ", C[i][j]);
}
printf("\n");
}
}

MPI_Finalize();
return 0;
}
```

output:

```
abhijeet@abhijeet-VirtualBox:~/MPI$ mpirun -np 4 ./9_2
Resulting Matrix C:
3.000000 8.000000 23.000000 68.000000
3.000000 8.000000 23.000000 68.000000
3.000000 8.000000 23.000000 68.000000
3.000000 8.000000 23.000000 68.000000
```