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

High Performance Computing Lab

Practical No. 9

---

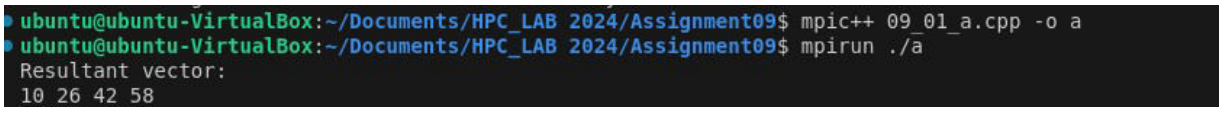
1. **Implement Matrix-Vector Multiplication using MPI. Use different number of processes and analyze the performance.**

Code:

```
#include <mpi.h>
#include <iostream>
#include <vector>

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 n = 4; // Matrix size (nxn) and Vector size
std::vector<int> matrix, vector(n), local_result(n / size);
std::vector<int> result(n);

if (rank == 0) {
matrix = {
1, 2, 3, 4,
5, 6, 7, 8,
9, 10, 11, 12,
13, 14, 15, 16
};

vector = {1, 1, 1, 1}; // Vector to multiply
}

// Broadcast the vector to all processes
MPI_Bcast(vector.data(), n, MPI_INT, 0, MPI_COMM_WORLD);

// Scatter matrix rows among processes
std::vector<int> local_matrix(n * (n / size));
MPI_Scatter(matrix.data(), n * (n / size), MPI_INT, local_matrix.data(),
```

```
n * (n / size), MPI_INT, 0, MPI_COMM_WORLD);

// Local matrix-vector multiplication
for (int i = 0; i < n / size; ++i) {
local_result[i] = 0;
for (int j = 0; j < n; ++j) {
local_result[i] += local_matrix[i * n + j] * vector[j];
}
}

// Gather the results
MPI_Gather(local_result.data(), n / size, MPI_INT, result.data(), n /
size, MPI_INT, 0, MPI_COMM_WORLD);

if (rank == 0) {
std::cout << "Resultant vector:" << std::endl;
for (int i = 0; i < n; ++i) {
std::cout << result[i] << " ";
}
std::cout << std::endl;
}

MPI_Finalize();
return 0;
}
```

Screenshot:



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

Code:

```
#include <mpi.h>
#include <iostream>
#include <vector>
```

```
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 n = 4; // Matrix size nxn
std::vector<int> A, B(n * n), C(n * n);
std::vector<int> local_A(n * (n / size)), local_C(n * (n / size));

if (rank == 0) {
A = {
1, 2, 3, 4,
5, 6, 7, 8,
9, 10, 11, 12,
13, 14, 15, 16
};

B = {
1, 1, 1, 1,
1, 1, 1, 1,
1, 1, 1, 1,
1, 1, 1, 1
};
}

// Broadcast matrix B to all processes
MPI_Bcast(B.data(), n * n, MPI_INT, 0, MPI_COMM_WORLD);

// Scatter rows of matrix A among processes
MPI_Scatter(A.data(), n * (n / size), MPI_INT, local_A.data(), n * (n /
size), MPI_INT, 0, MPI_COMM_WORLD);

// Local matrix-matrix multiplication
for (int i = 0; i < n / size; ++i) {
for (int j = 0; j < n; ++j) {
local_C[i * n + j] = 0;
for (int k = 0; k < n; ++k) {
local_C[i * n + j] += local_A[i * n + k] * B[k * n + j];
}
}
}

// Gather the result matrix C
MPI_Gather(local_C.data(), n * (n / size), MPI_INT, C.data(), n * (n /
size), MPI_INT, 0, MPI_COMM_WORLD);
```

```
if (rank == 0) {
std::cout << "Resultant matrix C:" << std::endl;
for (int i = 0; i < n; ++i) {
for (int j = 0; j < n; ++j) {
std::cout << C[i * n + j] << " ";
}
std::cout << std::endl;
}
}

MPI_Finalize();
return 0;
}
```

Screenshot: