Final Year B.Tech. (CSE) – VII [2024-25]

**6CS452: High Performance Computing Lab**

Assignment No: 10

**Analysis of MPI Programs**

**PRN:** 21510042  **Name:** Omkar Rajesh Auti

**Title:** Analysis of MPI Programs

**Problem Statement 1:**

Execute the MPI program (Program A) with a fixed size broadcast. Plot the performance of the broadcast with varying numbers of processes (with constant message size). Explain the performance observed.

Code for Creation of Sample Files:

import numpy as np

import os

def create\_matrix\_file(filename, rows=512, cols=512):

    # Create the directory if it doesn't exist

    directory = os.path.dirname(filename)

    if not os.path.exists(directory):

        os.makedirs(directory)

    # Generate a matrix of random floating-point numbers

    matrix = np.random.rand(rows, cols)

    # Write the matrix to the file

    with open(filename, 'w') as f:

        for row in matrix:

            row\_str = ' '.join(f'{val:.6f}' for val in row)

            f.write(row\_str + '\n')

# Create sample/in1 and sample/in2 files

create\_matrix\_file('sample/in1')

create\_matrix\_file('sample/in2')

print("Files sample/in1 and sample/in2 have been created if they did not exist.")

Corrected Code:

#include <assert.h>

#include <math.h>

#include <string.h>

#include <stdio.h>

#include <stdlib.h>

#include <mpi.h>

typedef struct

{

    float r;

    float i;

} complex;

static complex ctmp;

#define C\_SWAP(a, b) \

    {                \

        ctmp = (a);  \

        (a) = (b);   \

        (b) = ctmp;  \

    }

#define N 512

void c\_fft1d(complex \*r, int n, int isign)

{

    int m, i, i1, j, k, i2, l, l1, l2;

    float c1, c2, z;

    complex t, u;

    if (isign == 0)

        return;

    /\* Do the bit reversal \*/

    i2 = n >> 1;

    j = 0;

    for (i = 0; i < n - 1; i++)

    {

        if (i < j)

            C\_SWAP(r[i], r[j]);

        k = i2;

        while (k <= j)

        {

            j -= k;

            k >>= 1;

        }

        j += k;

    }

    /\* m = (int) log2((double)n); \*/

    for (i = n, m = 0; i > 1; m++, i /= 2)

        ;

    /\* Compute the FFT \*/

    c1 = -1.0;

    c2 = 0.0;

    l2 = 1;

    for (l = 0; l < m; l++)

    {

        l1 = l2;

        l2 <<= 1;

        u.r = 1.0;

        u.i = 0.0;

        for (j = 0; j < l1; j++)

        {

            for (i = j; i < n; i += l2)

            {

                i1 = i + l1;

                /\* t = u \* r[i1] \*/

                t.r = u.r \* r[i1].r - u.i \* r[i1].i;

                t.i = u.r \* r[i1].i + u.i \* r[i1].r;

                /\* r[i1] = r[i] - t \*/

                r[i1].r = r[i].r - t.r;

                r[i1].i = r[i].i - t.i;

                /\* r[i] = r[i] + t \*/

                r[i].r += t.r;

                r[i].i += t.i;

            }

            z = u.r \* c1 - u.i \* c2;

            u.i = u.r \* c2 + u.i \* c1;

            u.r = z;

        }

        c2 = sqrt((1.0 - c1) / 2.0);

        if (isign == -1) /\* FWD FFT \*/

            c2 = -c2;

        c1 = sqrt((1.0 + c1) / 2.0);

    }

    /\* Scaling for inverse transform \*/

    if (isign == 1)

    { /\* IFFT\*/

        for (i = 0; i < n; i++)

        {

            r[i].r /= n;

            r[i].i /= n;

        }

    }

}

void getData(char fileName[15], complex \*\*data)

{

    FILE \*fp = fopen(fileName, "r");

    int i, j, result;

    for (i = 0; i < N; i++)

    {

        for (j = 0; j < N; j++)

        {

            result = fscanf(fp, "%g", &data[i][j].r);

            data[i][j].i = 0.00;

        }

    }

    fclose(fp);

}

void transpose(complex \*\*data, complex \*\*transp)

{

    int i, j;

    for (i = 0; i < N; i++)

        for (j = 0; j < N; j++)

            transp[j][i] = data[i][j];

}

void mmpoint(complex \*\*data1, complex \*\*data2, complex \*\*data3)

{

    int i, j;

    float real, imag;

    for (i = 0; i < N; i++)

    {

        for (j = 0; j < N; j++)

        {

            data3[i][j].r = (data1[i][j].r \* data2[i][j].r) - (data1[i][j].i \* data2[i][j].i);

            data3[i][j].i = (data1[i][j].r \* data2[i][j].i) + (data1[i][j].i \* data2[i][j].r);

        }

    }

}

void printfile(char fileName[15], complex \*\*data)

{

    FILE \*fp = fopen(fileName, "w");

    int i, j;

    for (i = 0; i < N; i++)

    {

        for (j = 0; j < N; j++)

        {

            fprintf(fp, "   %.7e", data[i][j].r);

        }

        fprintf(fp, "\n");

    }

    fclose(fp);

}

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

{

    int my\_rank, p, source = 0, dest, x;

    complex \*\*data1, \*\*data2, \*\*data3, \*\*data4;

    data1 = malloc(N \* sizeof(complex \*));

    data2 = malloc(N \* sizeof(complex \*));

    data3 = malloc(N \* sizeof(complex \*));

    data4 = malloc(N \* sizeof(complex \*));

    for (x = 0; x < N; x++)

    {

        data1[x] = malloc(N \* sizeof(complex));

        data2[x] = malloc(N \* sizeof(complex));

        data3[x] = malloc(N \* sizeof(complex));

        data4[x] = malloc(N \* sizeof(complex));

    }

    complex \*vec;

    char fileName1[15] = "sample/in1";

    char fileName2[15] = "sample/in2";

    char fileName3[15] = "mpi\_out\_test";

    MPI\_Status status;

    MPI\_Init(&argc, &argv);

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

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

    /\* Setup description of the 4 MPI\_FLOAT fields x, y, z, velocity \*/

    int blocklens[2] = {1, 1};

    MPI\_Aint indices[2] = {0, sizeof(float)};

    MPI\_Datatype old\_types[2] = {MPI\_FLOAT, MPI\_FLOAT};

    MPI\_Datatype mystruct;

    /\* Make relative \*/

    // MPI\_Type\_struct(2, blocklens, indices, old\_types, &mystruct);

    MPI\_Type\_create\_struct(2, blocklens, indices, old\_types, &mystruct);

    MPI\_Type\_commit(&mystruct);

    int i, j;

    double startTime, stopTime;

    // Starting and send rows of data1, data2

    int offset;

    int tag = 345;

    int rows = N / p;

    int lb = my\_rank \* rows;

    int hb = lb + rows;

    printf("%d have lb = %d and hb = %d\n", my\_rank, lb, hb);

    // Starting and send rows of data1, data2

    if (my\_rank == 0)

    {

        getData(fileName1, data1);

        getData(fileName2, data2);

        /\* Start Clock \*/

        printf("\nStarting clock.\n");

        startTime = MPI\_Wtime();

        for (i = 1; i < p; i++)

        {

            offset = i \* rows;

            for (j = offset; j < (offset + rows); j++)

            {

                MPI\_Send(&data1[j][0], N, mystruct, i, tag, MPI\_COMM\_WORLD);

                MPI\_Send(&data2[j][0], N, mystruct, i, tag, MPI\_COMM\_WORLD);

            }

        }

    }

    else

    {

        for (j = lb; j < hb; j++)

        {

            MPI\_Recv(data1[j], N, mystruct, 0, tag, MPI\_COMM\_WORLD, &status);

            MPI\_Recv(data2[j], N, mystruct, 0, tag, MPI\_COMM\_WORLD, &status);

        }

    }

    // Doing fft1d forward for data1 and data2 rows

    vec = (complex \*)malloc(N \* sizeof(complex));

    for (i = lb; i < hb; i++)

    {

        for (j = 0; j < N; j++)

        {

            vec[j] = data1[i][j];

        }

        c\_fft1d(vec, N, -1);

        for (j = 0; j < N; j++)

        {

            data1[i][j] = vec[j];

        }

    }

    free(vec);

    vec = (complex \*)malloc(N \* sizeof(complex));

    for (i = lb; i < hb; i++)

    {

        for (j = 0; j < N; j++)

        {

            vec[j] = data2[i][j];

        }

        c\_fft1d(vec, N, -1);

        for (j = 0; j < N; j++)

        {

            data2[i][j] = vec[j];

        }

    }

    free(vec);

    // Receving rows of data1, data2

    if (my\_rank == 0)

    {

        for (i = 1; i < p; i++)

        {

            offset = i \* rows;

            for (j = offset; j < (offset + rows); j++)

            {

                MPI\_Recv(data1[j], N, mystruct, i, tag, MPI\_COMM\_WORLD, &status);

                MPI\_Recv(data2[j], N, mystruct, i, tag, MPI\_COMM\_WORLD, &status);

            }

        }

    }

    else

    {

        for (j = lb; j < hb; j++)

        {

            MPI\_Send(&data1[j][0], N, mystruct, 0, tag, MPI\_COMM\_WORLD);

            MPI\_Send(&data2[j][0], N, mystruct, 0, tag, MPI\_COMM\_WORLD);

        }

    }

    // Starting and send columns of data1, data2

    if (my\_rank == 0)

    {

        transpose(data1, data3);

        transpose(data2, data4);

        for (i = 1; i < p; i++)

        {

            offset = i \* rows;

            for (j = offset; j < (offset + rows); j++)

            {

                MPI\_Send(&data3[j][0], N, mystruct, i, tag, MPI\_COMM\_WORLD);

                MPI\_Send(&data4[j][0], N, mystruct, i, tag, MPI\_COMM\_WORLD);

            }

        }

    }

    else

    {

        for (j = lb; j < hb; j++)

        {

            MPI\_Recv(data3[j], N, mystruct, 0, tag, MPI\_COMM\_WORLD, &status);

            MPI\_Recv(data4[j], N, mystruct, 0, tag, MPI\_COMM\_WORLD, &status);

        }

    }

    // Doing fft1d forward for data1 and data2 columns

    vec = (complex \*)malloc(N \* sizeof(complex));

    for (i = lb; i < hb; i++)

    {

        for (j = 0; j < N; j++)

        {

            vec[j] = data3[i][j];

        }

        c\_fft1d(vec, N, -1);

        for (j = 0; j < N; j++)

        {

            data3[i][j] = vec[j];

        }

    }

    free(vec);

    vec = (complex \*)malloc(N \* sizeof(complex));

    for (i = lb; i < hb; i++)

    {

        for (j = 0; j < N; j++)

        {

            vec[j] = data4[i][j];

        }

        c\_fft1d(vec, N, -1);

        for (j = 0; j < N; j++)

        {

            data4[i][j] = vec[j];

        }

    }

    free(vec);

    // Receving columns of data1, data2

    if (my\_rank == 0)

    {

        for (i = 1; i < p; i++)

        {

            offset = i \* rows;

            for (j = offset; j < (offset + rows); j++)

            {

                MPI\_Recv(data3[j], N, mystruct, i, tag, MPI\_COMM\_WORLD, &status);

                MPI\_Recv(data4[j], N, mystruct, i, tag, MPI\_COMM\_WORLD, &status);

            }

        }

    }

    else

    {

        for (j = lb; j < hb; j++)

        {

            MPI\_Send(&data3[j][0], N, mystruct, 0, tag, MPI\_COMM\_WORLD);

            MPI\_Send(&data4[j][0], N, mystruct, 0, tag, MPI\_COMM\_WORLD);

        }

    }

    if (my\_rank == 0)

    {

        transpose(data3, data1);

        transpose(data4, data2);

        mmpoint(data1, data2, data3);

    }

    // Starting and send rows of data1, data2

    if (my\_rank == 0)

    {

        for (i = 1; i < p; i++)

        {

            offset = i \* rows;

            for (j = offset; j < (offset + rows); j++)

            {

                MPI\_Send(&data3[j][0], N, mystruct, i, tag, MPI\_COMM\_WORLD);

            }

        }

    }

    else

    {

        for (j = lb; j < hb; j++)

        {

            MPI\_Recv(data3[j], N, mystruct, 0, tag, MPI\_COMM\_WORLD, &status);

        }

    }

    // Doing fft1d forward for data1 and data2 rows

    vec = (complex \*)malloc(N \* sizeof(complex));

    for (i = lb; i < hb; i++)

    {

        for (j = 0; j < N; j++)

        {

            vec[j] = data3[i][j];

        }

        c\_fft1d(vec, N, 1);

        for (j = 0; j < N; j++)

        {

            data3[i][j] = vec[j];

        }

    }

    free(vec);

    // Receving rows of data1, data2

    if (my\_rank == 0)

    {

        for (i = 1; i < p; i++)

        {

            offset = i \* rows;

            for (j = offset; j < (offset + rows); j++)

            {

                MPI\_Recv(data3[j], N, mystruct, i, tag, MPI\_COMM\_WORLD, &status);

            }

        }

    }

    else

    {

        for (j = lb; j < hb; j++)

        {

            MPI\_Send(&data3[j][0], N, mystruct, 0, tag, MPI\_COMM\_WORLD);

        }

    }

    // Starting and send columns of data1, data2

    if (my\_rank == 0)

    {

        transpose(data3, data4);

        for (i = 1; i < p; i++)

        {

            offset = i \* rows;

            for (j = offset; j < (offset + rows); j++)

            {

                MPI\_Send(&data4[j][0], N, mystruct, i, tag, MPI\_COMM\_WORLD);

            }

        }

    }

    else

    {

        for (j = lb; j < hb; j++)

        {

            MPI\_Recv(data4[j], N, mystruct, 0, tag, MPI\_COMM\_WORLD, &status);

        }

    }

    // Doing fft1d forward for data1 and data2 columns

    vec = (complex \*)malloc(N \* sizeof(complex));

    for (i = lb; i < hb; i++)

    {

        for (j = 0; j < N; j++)

        {

            vec[j] = data4[i][j];

        }

        c\_fft1d(vec, N, 1);

        for (j = 0; j < N; j++)

        {

            data4[i][j] = vec[j];

        }

    }

    free(vec);

    // Receving columns of data1, data2

    if (my\_rank == 0)

    {

        for (i = 1; i < p; i++)

        {

            offset = i \* rows;

            for (j = offset; j < (offset + rows); j++)

            {

                MPI\_Recv(data4[j], N, mystruct, i, tag, MPI\_COMM\_WORLD, &status);

            }

        }

    }

    else

    {

        for (j = lb; j < hb; j++)

        {

            MPI\_Send(&data4[j][0], N, mystruct, 0, tag, MPI\_COMM\_WORLD);

        }

    }

    if (my\_rank == 0)

    {

        transpose(data4, data3);

        /\* Stop Clock \*/

        stopTime = MPI\_Wtime();

        printf("\nElapsed time = %lf s.\n", (stopTime - startTime));

        printf("--------------------------------------------\n");

    }

    MPI\_Finalize();

    if (my\_rank == 0)

    {

        printfile(fileName3, data3);

    }

    free(data1);

    free(data2);

    free(data3);

    free(data4);

    return 0;

}

**Output:**

**Problem Statement 2:**

Repeat problem 2 above with varying message sizes for reduction (Program B). Explain the observed performance of the reduction operation.