Mahendra GHarge
2018BTECS 00033
HPC Lab

Assignment 6

Q1. Study the gather and scatter functions of MPI.

Sol: MPI_Scatter:MPI_Scatter is a collective routine that is very similar to MPI_Bcast .MPI_Scatter involves a designated root process sending data to all processes in a communicator. The primary difference between MPI_Bcast and MPI_Scatter is small but important. MPI_Bcast sends the same piece of data to all processes while MPI_Scatter sends chunks of an array to different processes. Check out the illustration below for further clarification.

MPI_Gather: MPI_Gather is the inverse of MPI_Scatter. Instead of spreading elements from one process to many processes, MPI_Gather takes elements from many processes and gathers them to one single process. This routine is highly useful to many parallel algorithms, such as parallel sorting and searching. Below is a simple illustration of this algorithm.

Q2. Consider an implementation of the gather operation given to you (Program E) – in this implementation, each process sends its message to process 0, which gathers the message. Compare this code to the one that uses the MPI gather operation (Program F). Compare the performance for the fixed message size (1K words at each process) case with varying number of processes (1 - 16). Which implementation is better? Why?

E.C

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <mpi.h>
int main(int argc, char *argv[])
{
    if (argc != 2)
    {
        printf("Usage : gather message_size\n");
        return 1;
    }
    int rank;
    int num_procs;
    int size = atoi(argv[1]);
    char input_buffer[size];
    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &num_procs);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    int i;
    char recv_buffer[size * num_procs];
    srand(time(NULL));
    for (i = 0; i < size; i++)
        input_buffer[i] = rand() % 256;
    double total_time = 0.0;
```

```
    double start_time = 0.0;
    for (i = 0; i < 100; i++)
    {
        MPI_Barrier(MPI_COMM_WORLD);
        start_time = MPI_Wtime();

        MPI_Send(input_buffer, size, MPI_CHAR, 0, 99, MPI_COMM_WORLD);
        if (rank == 0)
        {
            int j;
            for (j = 0; j < num_procs; j++)
            {
                MPI_Recv(recv_buffer +
                                j * size,
                         size, MPI_CHAR, j, 99, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);
            }
        }
        MPI_Barrier(MPI_COMM_WORLD);
        total_time += (MPI_Wtime() - start_time);
    }
    if (rank == 0)
    {
        printf("Average time for gather : %f \n", total_time / 100);
    }
    MPI_Finalize();
}
```

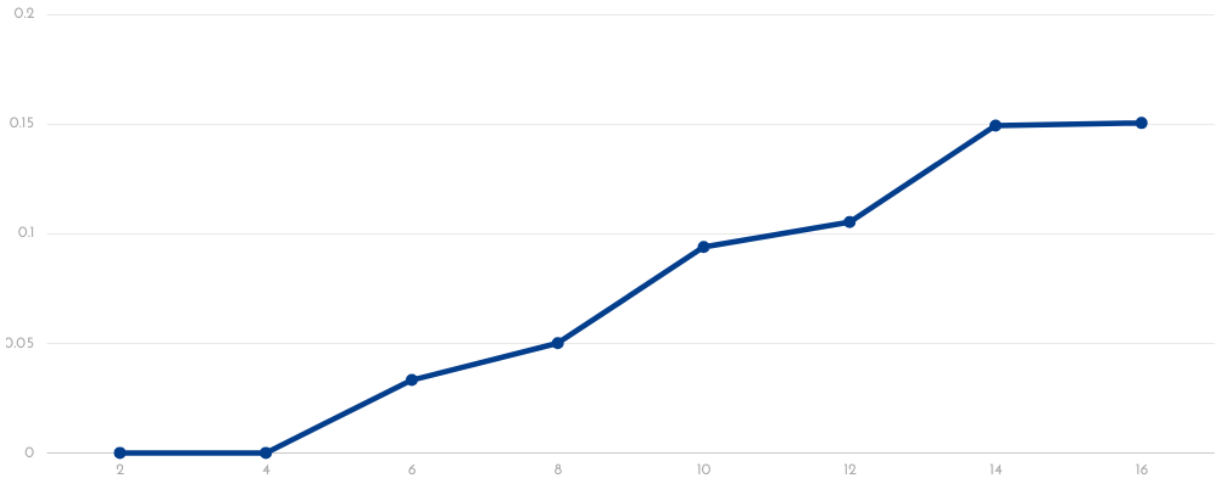
F.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <mpi.h>
int main(int argc, char *argv[])
{
    if (argc != 2)
    {
        printf("Usage : gather message_size\n");
        return 1;
    }
    int rank;
    int num_procs;
    int size = atoi(argv[1]);
    char input_buffer[size];
    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &num_procs);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    int i;
    char recv_buffer[size * num_procs];
    srand(time(NULL));
    for (i = 0; i < size; i++)
        input_buffer[i] = rand() % 256;
    double total_time = 0.0;
    double start_time = 0.0;
    for (i = 0; i < 100; i++)
    {
        MPI_Barrier(MPI_COMM_WORLD);
        start_time = MPI_Wtime();

        MPI_Gather(input_buffer, size, MPI_CHAR, recv_buffer, size,
MPI_CHAR, 0, MPI_COMM_WORLD);
        MPI_Barrier(MPI_COMM_WORLD);
        total_time += (MPI_Wtime() - start_time);
    }
    if (rank == 0)
    {
```

```
        printf("Average time for gather : %f secs\n", total_time / 100);
    }
    MPI_Finalize();
}
```

Sol:



The programm E.c has some execution errors. Above is the line graph for program f.c with fixed message size and varying number of processes from 2 to 16.

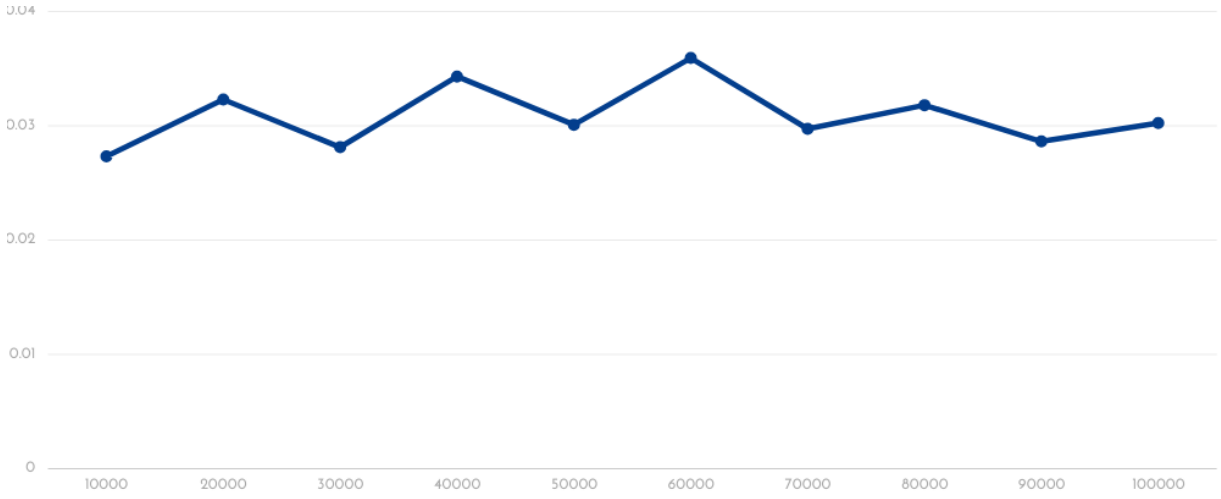
Q3. Run the scatter operation (Program G) with varying message sizes (10K to 100K), with a fixed number of processes (8). Plot the runtime as a function of the message size. Explain the observed performance

G.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <mpi.h>
int main(int argc, char *argv[])
{
    if (argc != 2)
    {
        printf("Usage : scatter message_size\n");
        return 1;
    }
    int rank;
    int num_procs;
    int size = atoi(argv[1]);
    char input_buffer[size];
    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &num_procs);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    int i;
    char recv_buffer[size / num_procs];
    srand(time(NULL));
    for (i = 0; i < size; i++)
        input_buffer[i] = rand() % 256;
    double total_time = 0.0;
    double start_time = 0.0;
    for (i = 0; i < 100; i++)
    {
        MPI_Barrier(MPI_COMM_WORLD);
        start_time = MPI_Wtime();
```

```
        MPI_Scatter(input_buffer, size / num_procs, MPI_CHAR, recv_buffer,
size / num_procs, MPI_CHAR, 0, MPI_COMM_WORLD);
        MPI_Barrier(MPI_COMM_WORLD);
        total_time += (MPI_Wtime() - start_time);
    }
    if (rank == 0)
    {
        printf("Average time for scatter : %f secs\n", total_time / 100);
    }
    MPI_Finalize();
}
```

SOL:



As the message size increases, the scatter operation does not show a regular trend but has a zigzag fashion curve.