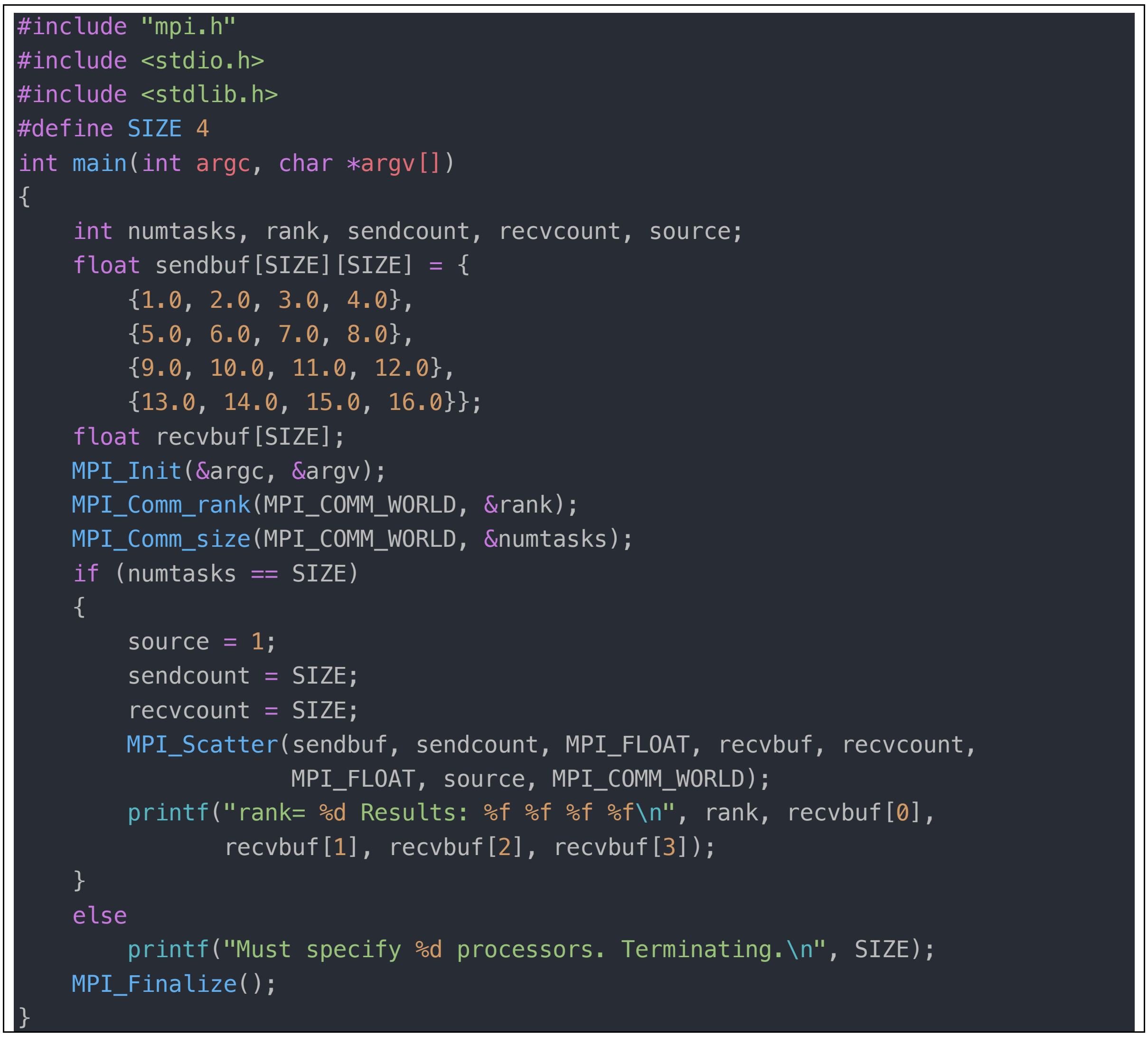
NAME	JACOB JOHN
REGISTER NO.	16BCE2205
E-MAIL	jacob.john2016@vitstudent.ac.in

LAB ASSESSMENT #10

SCENARIO – 1: Collective Communication

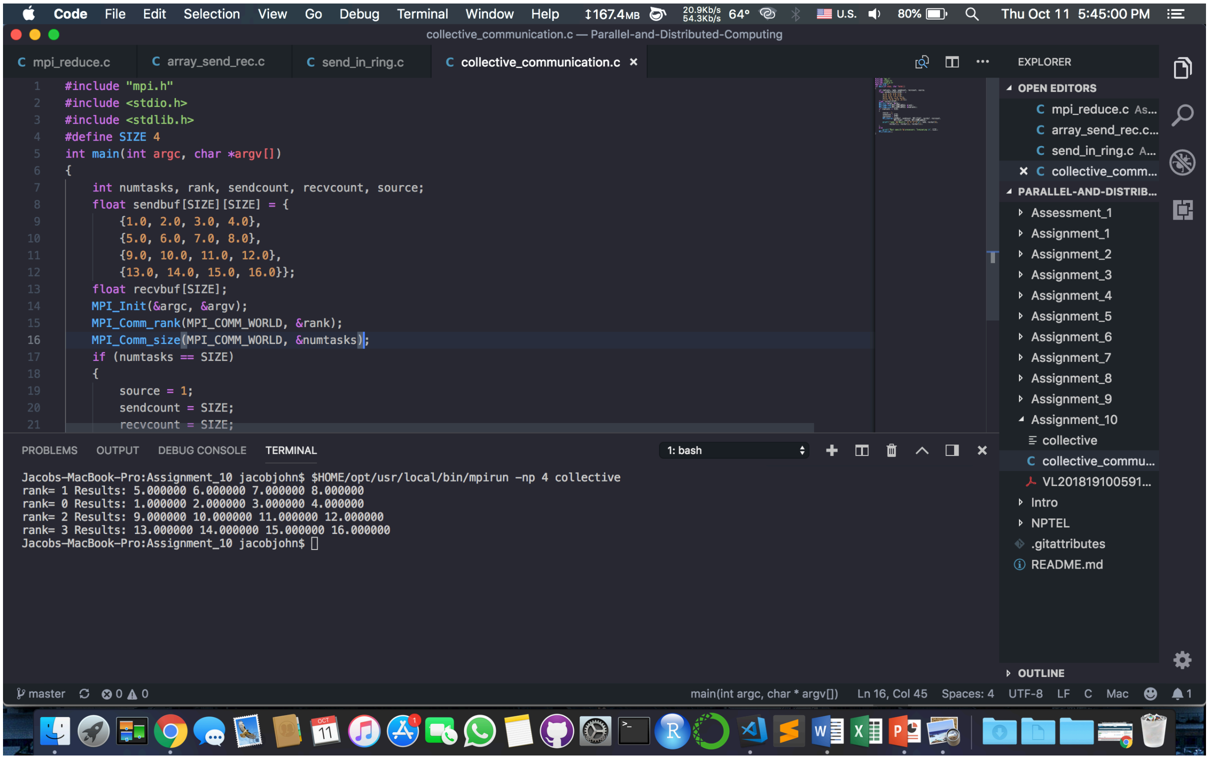
Study the given C program that takes an array of elements and distributes the elements in the order of process rank. The first element (in red) goes to process zero, the second element (in green) goes to process one, and so on. Although the root process (process zero) contains the entire array of data, *MPI_Scatter* will copy the appropriate element into the receiving buffer of the process. Analyze its key factors in terms of network application system. Assume that the data consists of a single integer. Process zero reads the data from the user.

Code:



```c
#include "mpi.h"
#include <stdio.h>
#include <stdlib.h>
#define SIZE 4
int main(int argc, char *argv[])
{
    int numtasks, rank, sendcount, recvcount, source;
    float sendbuf[SIZE][SIZE] = {
        {1.0, 2.0, 3.0, 4.0},
        {5.0, 6.0, 7.0, 8.0},
        {9.0, 10.0, 11.0, 12.0},
        {13.0, 14.0, 15.0, 16.0}};
    float recvbuf[SIZE];
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &numtasks);
    if (numtasks == SIZE)
    {
        source = 1;
        sendcount = SIZE;
        recvcount = SIZE;
        MPI_Scatter(sendbuf, sendcount, MPI_FLOAT, recvbuf, recvcount,
                    MPI_FLOAT, source, MPI_COMM_WORLD);
        printf("rank= %d Results: %f %f %f %f\n", rank, recvbuf[0],
               recvbuf[1], recvbuf[2], recvbuf[3]);
    }
    else
        printf("Must specify %d processors. Terminating.\n", SIZE);
    MPI_Finalize();
}
```

Output:



```c
#include "mpi.h"
#include <stdio.h>
#include <stdlib.h>
#define SIZE 4
int main(int argc, char *argv[])
{
    int numtasks, rank, sendcount, recvcount, source;
    float sendbuf[SIZE][SIZE] = {
        {1.0, 2.0, 3.0, 4.0},
        {5.0, 6.0, 7.0, 8.0},
        {9.0, 10.0, 11.0, 12.0},
        {13.0, 14.0, 15.0, 16.0}};
    float recvbuf[SIZE];
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &numtasks);
    if (numtasks == SIZE)
    {
        source = 1;
        sendcount = SIZE;
        recvcount = SIZE;
```

```
Jacobs-MacBook-Pro:Assignment_10 jacobjohn$ $HOME/opt/usr/local/bin/mpirun -np 4 collective
rank= 1 Results: 5.000000 6.000000 7.000000 8.000000
rank= 0 Results: 1.000000 2.000000 3.000000 4.000000
rank= 2 Results: 9.000000 10.000000 11.000000 12.000000
rank= 3 Results: 13.000000 14.000000 15.000000 16.000000
Jacobs-MacBook-Pro:Assignment_10 jacobjohn$
```

Execution

```
Jacobs-MacBook-Pro:Assignment_10 jacobjohn$ $HOME/opt/usr/local/bin/mpirun -np 4 collective
rank= 1 Results: 5.000000 6.000000 7.000000 8.000000
rank= 0 Results: 1.000000 2.000000 3.000000 4.000000
rank= 2 Results: 9.000000 10.000000 11.000000 12.000000
rank= 3 Results: 13.000000 14.000000 15.000000 16.000000
```

Conceptual discussion

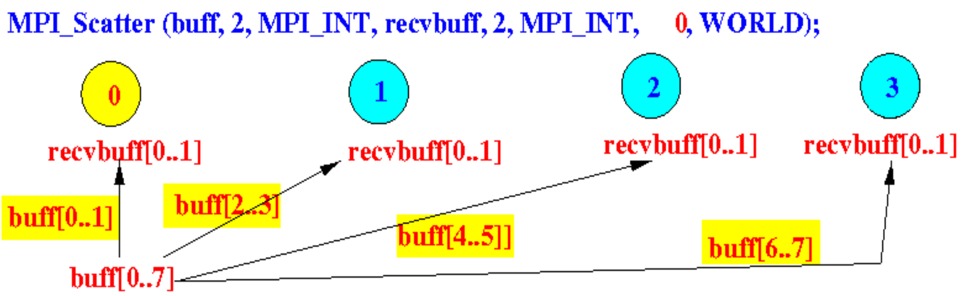
The MPI_Scatter() is used to split an array into N parts and send one of the parts to each MPI process.

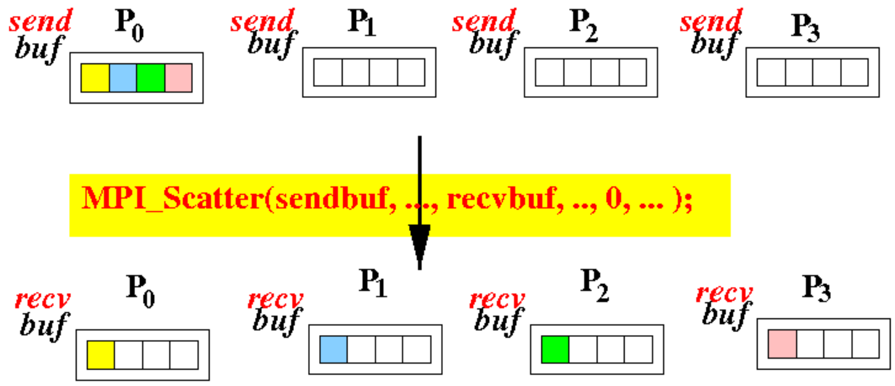
Syntax of the MPI_Scatter() call:

```
MPI_Scatter(void *sendbuf,          // Distribute sendbuf evenly to recvbuf
            int sendcount,          // # items sent to EACH processor
            MPI_Datatype sendtype,

            void* recvbuf,
            int recvcount,
            MPI_DATATYPE recvtype,

            int rootID,             // Sending Processor!
            MPI_Comm comm)
```

- sendbuf – the data (sendbuf in the rootID processor must have a valid data)

- sendcount – number of items sent to each processor (valid for rootID only)
- sendtype – type of data sent (valid for rootID only)
- recvbuf – buffer for receiving data
- recvcount – number of items to receive
- rootID – id of root processor (who is doing the send operation)
- comm – the communicator group





In the program, since the number of processors running are 4 and the size is defined as 4, the array is divided among 4 processors. Each processor prints the part of the array as according to its rank. Even the rank 0 processor receives and prints the data in this case.

```
MPI_Scatter(sendbuf, sendcount = 4,                //4 items will be sent
                 MPI_FLOAT, recvbuf, recvcount = 4, //4 items will be received
                 MPI_FLOAT, source = 1,              //root processor sending is 1
                 MPI_COMM_WORLD);
```

Since print isn't an atomic operation, the print doesn't happen in order. However, each array gets its part of the array in an orderly fashion due to *MPI_Scatter's* ability to divide the array into 4 parts.