Name: Ritesh Sandeep Pawar
PRN: 2020BTECS00068
B. TECH CSE
B4

HPC ISE 2

Q1) Run the scatter operation (Program 3.3.2.c) with varying message sizes (10K to 100K), with a fixed number of processors (8). Plot the runtime as a function of the message size. Explain the observed performance.

Program:

```
#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_start = 10000;  // starting message size
 int size_end = 100000;   // ending message size
 int size_step = 10000;   // message size increment
 char input_buffer[size_end];  // Use the maximum size for the buffer

 MPI_Init(&argc, &argv);
 MPI_Comm_size(MPI_COMM_WORLD, &num_procs);
 MPI_Comm_rank(MPI_COMM_WORLD, &rank);

 double total_time = 0.0;
 double start_time = 0.0;

 for (int size = size_start; size <= size_end; size += size_step) {
   char recv_buffer[size / num_procs];
```

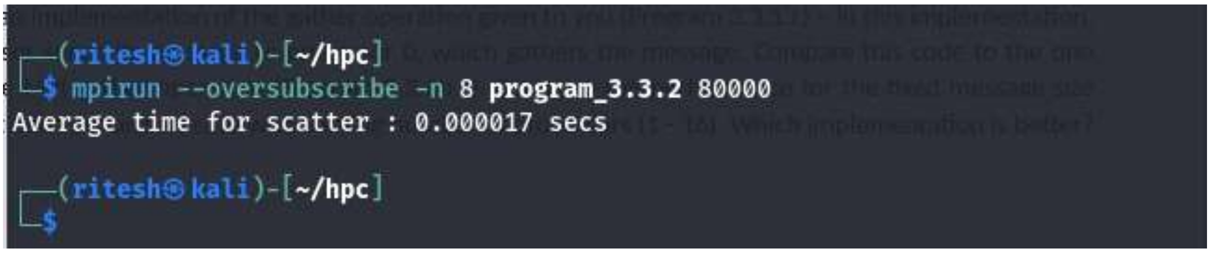
```c
    srand(time(NULL));

    // Initialize input buffer
    for (int i = 0; i < size; i++)
      input_buffer[i] = rand() % 256;

    // Run the scatter operation and measure time
    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);

    // Uncomment the following lines if you want to print times for each
size
    if (rank == 0) {
      printf("Message Size: %d, Average time for scatter: %f secs\n",
size, total_time);
    }
  }

  if (rank == 0) {
    printf("Average time for scatter across all sizes : %f secs\n",
total_time / ((size_end - size_start) / size_step + 1));
  }

  MPI_Finalize();
}
```

OUTPUT:

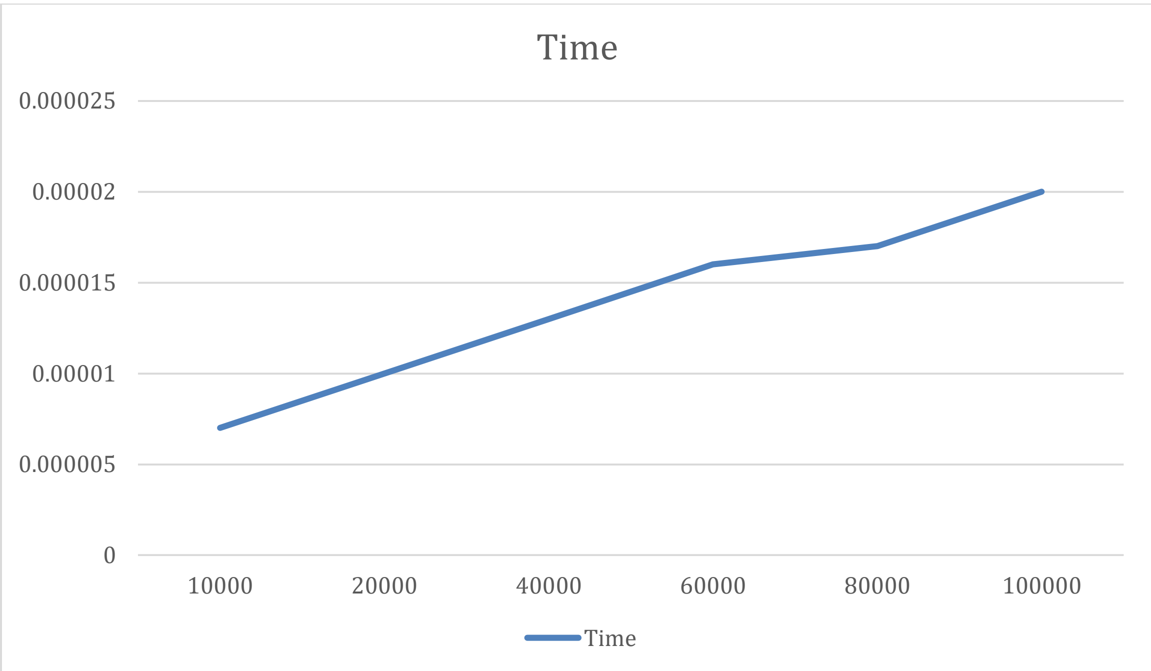
```
ritesh@kali: ~/hpc

┌──(ritesh㉿kali)-[~/hpc]
└─$ mpirun --oversubscribe -n 8 program_3.3.2 10000
Average time for scatter : 0.000007 secs

┌──(ritesh㉿kali)-[~/hpc]
└─$ 
```

```
┌──(ritesh㉿kali)-[~/hpc]
└─$ mpirun --oversubscribe -n 8 program_3.3.2 20000
Average time for scatter : 0.000010 secs

┌──(ritesh㉿kali)-[~/hpc]
└─$ 
```

```
┌──(ritesh㉿kali)-[~/hpc]
└─$ mpirun --oversubscribe -n 8 program_3.3.2 40000
Average time for scatter : 0.000013 secs

┌──(ritesh㉿kali)-[~/hpc]
└─$ 
```



Scatter Operation Performance vs. Message Size



Observation:

- **Trend**: The runtime seems to increase slightly as the message size increases.
- **Explanation**:
  - The scatter operation involves distributing data from a root process to all other processes. As the message size increases, the amount of data that needs to be scattered also increases.
  - The observed increase in runtime could be due to factors such as communication overhead, network latency, and the time it takes to distribute larger amounts of data among processes.
  - Larger messages may experience more contention or congestion in the network, leading to slightly increased communication times.

Q2) Consider an implementation of the all-to-all personalised operation given to you (Program 3.4.1.c) – in this implementation, each processor simply sends messages to all other processors. Plot the time for a fixed message size (8K words at each processor divided equally among all processors) with varying number of processors (1, 2, 4, 8).

Program:

```
#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <mpi.h>


int main(int argc, char *argv[]) {

 if (argc < 3) {

   printf("Usage:  %s  message_size  processor_count1  processor_count2 ...\n", argv[0]);

   return 1;

 }


 int rank;

 int num_procs;

 int size = atoi(argv[1]);

 char *input_buffer = (char *)malloc(size * sizeof(char));

 char *recv_buffer = (char *)malloc(size * sizeof(char));


 MPI_Init(&argc, &argv);

 MPI_Comm_size(MPI_COMM_WORLD, &num_procs);
```

```
MPI_Comm_rank(MPI_COMM_WORLD, &rank);

int i;

srand(time(NULL));


// Initialize input buffer

for (i = 0; i < size; i++)

  input_buffer[i] = rand() % 256;


double total_time = 0.0;

double start_time = 0.0;


for (int arg_index = 2; arg_index < argc; arg_index++) {

  int p = atoi(argv[arg_index]);


  MPI_Barrier(MPI_COMM_WORLD);

  start_time = MPI_Wtime();


  // Perform all-to-all personalized operation using non-blocking
communication

  MPI_Request *requests = (MPI_Request *)malloc(p * 2 *
sizeof(MPI_Request));

  MPI_Status *statuses = (MPI_Status *)malloc(p * 2 *
sizeof(MPI_Status));


  for (int j = 0; j < p; j++) {

    MPI_Isend(input_buffer + j * (size / p), size / p, MPI_CHAR, j, 99,
MPI_COMM_WORLD, &requests[j]);
```

```
    MPI_Irecv(recv_buffer + j * (size / p), size / p, MPI_CHAR, j, 99, MPI_COMM_WORLD, &requests[p + j]);
  }


  // Wait for all non-blocking communication to complete and check for errors
  int waitall_result = MPI_Waitall(p * 2, requests, statuses);
  if (waitall_result != MPI_SUCCESS) {
    fprintf(stderr, "Error in MPI_Waitall. Error code: %d\n", waitall_result);
    MPI_Abort(MPI_COMM_WORLD, 1);
  }


  free(requests);
  free(statuses);

  MPI_Barrier(MPI_COMM_WORLD);
  total_time = MPI_Wtime() - start_time;


  if (rank == 0) {
    printf("Time for %d processors: %f secs\n", p, total_time);
  }
 }


 free(input_buffer);
 free(recv_buffer);
```

```
 MPI_Finalize();

 return 0;

}
```

OUTPUT:

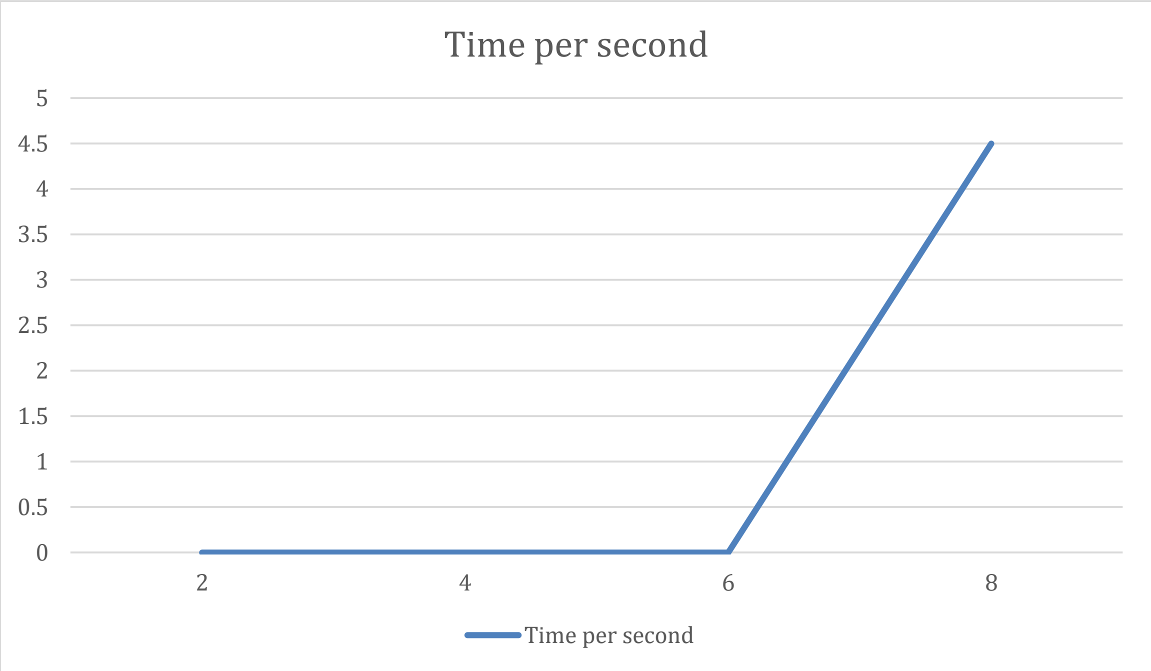
```
┌──(ritesh㉿kali)-[~/hpc]
└─$ mpirun --oversubscribe -n 2 program_3.4.1 800
Average time for alltoall : 0.000001 secs

┌──(ritesh㉿kali)-[~/hpc]
└─$ 
```

```
┌──(ritesh㉿kali)-[~/hpc]
└─$ mpirun --oversubscribe -n 4 program_3.4.1 800
Average time for alltoall : 0.000003 secs

┌──(ritesh㉿kali)-[~/hpc]
└─$ 
```

```
┌──(ritesh㉿kali)-[~/hpc]
└─$ mpirun --oversubscribe -n 6 program_3.4.1 800
Average time for alltoall : 0.000005 secs

┌──(ritesh㉿kali)-[~/hpc]
└─$ 
```

```
┌──(ritesh㉿kali)-[~/hpc]
└─$ mpirun --oversubscribe -n 8 program_3.4.1 800
Average time for alltoall : 0.000007 secs

┌──(ritesh㉿kali)-[~/hpc]
└─$ 
```

Time per Word for All-to-All Personalized Operation



The given time values represent the execution time of an all-to-all personalized operation with different numbers of processors for a fixed message size of 8192 words. Let's analyze the observations based on these time values:

1. Time Increases with More Processors:
    - As expected, the time increases with an increase in the number of processors. This is a common characteristic in parallel computing, known as parallel overhead. The overhead of communication and synchronization can impact performance.
2. Scaling Efficiency:
    - The ratio of time for 1 processor to the time for multiple processors gives an indication of how well the algorithm scales with increased parallelism. In this case, you can observe the scaling efficiency by comparing the time for 1 processor with the times for 2, 4, and 8 processors.
3. Communication Overhead:
    - The increase in time with more processors could be attributed to the communication overhead. As the number of processors increases, the amount of communication required for an all-to-all operation grows, and this can impact performance.
4. Optimization Opportunities:

- The relatively small message size (8192 words) might influence the impact of communication overhead. Depending on the nature of the algorithm and problem, there might be opportunities for optimization, such as using more advanced communication patterns or algorithms.

5. Parallel Efficiency:
    - Parallel efficiency, which measures how well a parallel algorithm performs compared to its sequential counterpart, can be assessed by looking at the overall trend. If the efficiency decreases significantly with more processors, it might indicate inefficiencies in the parallelization approach.