# CS546 Parallel and distributed Processing

## Programming Assignment

## Design

In this assignment, we have to implement a simple MPI-IO program. The homework has two parts.

1. One is to simply gain some familiarity with MPI-IO. MPI-IO will likely be part of whatever I/O stack you will be using in an HPC environment. We want each MPI process to write its rank to a common file using The MPI-IO interface.

Basically a parallel I/O system for distributed memory architectures will need a mechanism to specify collective operations and specify noncontiguous data layout in memory and file .Reading and writing in parallel is like receiving and sending messages. Hence, an MPI-like machinery is a good setting for Parallel I/O

In the first part, we want each MPI process to write its rank to a common file. In command line line argument, it will read N (number of processes), Output file name to print rank of each process. Here, each process participates in writing a portion of a common file.

Initially we have declare file pointer using :MPI_File fh.I used the MPI functions for writing must be

Preceded by a call to MPI_File_open to open a file.

int MPI_File_open(MPI_Comm comm, char *filename,int amode, MPI_Info info, MPI_File *fh).

Then, While opening the file in the write mode, use the appropriate flag/s in MPI_File_open: MPI_MODE_WRONLY for writing.

To write the rank to a common file, we need to calculate the displacement i.e number of bytes to skip from the start of the file. I've taken size of buffer is 4. and calculated offset for each rank and stored offset,rank into buffer as follows.

```
buf[0] =offset;
buf[1] =rank;
buf[2] =0;
buf[3] =0;
```

I've used MPI_File_write_at to writedata as int MPI_File_write_at(MPI_File fh, MPI_Offset offset,void *buf, int count, MPI_Datatype datatype,MPI_Status *status). And Close the file using: MPI_File_close(MPI_File fh). Moreover, I added error handling to all the MPI Functions. If error occurs,it will throw an error.

Sample output : `od -td -v abc.txt`

```
0000000           0           0           0           0
0000020          16           1           0           0
0000040          32           2           0           0
0000060          48           3           0           0
0000100          64           4           0           0
0000120          80           5           0           0
```

```
0000140          96           6           0           0
0000160         112           7           0           0
0000200         128           8           0           0
0000220         144           9           0           0
0000240         160          10           0           0
0000260         176          11           0           0
0000300         192          12           0           0
0000320         208          13           0           0
0000340         224          14           0           0
0000360         240          15           0           0
0000400
```

2. The second part is to build a parallel I/O benchmark using MPI-IO. In this Benchmark, we will open a temporary file, write random data to a file, and close the file. Then we will open it again, read the data from the file and close it again. All operations are performed by each rank to a common file.

Basically a parallel I/O system for distributed memory architectures will need a mechanism to specify collective operations and specify noncontiguous data layout in memory and file .Reading and writing in parallel is like receiving and sending messages. Hence, an MPI-like machinery is a good setting for Parallel I/O

In the second part, we have to build a parallel I/O benchmark using MPI-IO. we will open a temporary file, write random data to a file, and close the file. Then we will open it again, read the data from the file and close it again. All operations are performed by each rank to a common file.At the end,we have delete the file.

Initially read the arguments number of processes, outputfile to print timings, print ranks and MB for each rank. we need to open the file and write random data as follows: Firstly, we have declare file pointer using :MPI_File fh.I used the MPI functions for writing must be preceded by a call to MPI_File_open to open a file.

int MPI_File_open(MPI_Comm comm, char *filename,int amode, MPI_Info info, MPI_File *fh).

Then, While opening the file in the write mode, use the appropriate flag/s in MPI_File_open: MPI_MODE_WRONLY for writing.

I've assigned 1MB(long N=5152*512/2) and calculated total size for each rank as follows : N=N*p.where p is Mb for each rank. Based on I have written random data of size in to buffer and calculated offset to displacement.
Then,I've used MPI_File_write_at to writedata as int MPI_File_write_at(MPI_File fh, MPI_Offset offset,void *buf, int count, MPI_Datatype datatype,MPI_Status *status). And Close the file using: MPI_File_close(MPI_File fh). Moreover, I added error handling to all the MPI Functions. If error occurs,it will throw an error.

Now,I have to open the file and read the data form a file which we written data. .I used the MPI functions for writing must be preceded by a call to MPI_File_open to open a file.

int MPI_File_open(MPI_Comm comm, char *filename,int amode, MPI_Info info, MPI_File *fh).

Then, While opening the file in the read mode, use the appropriate flag/s in MPI_File_open: MPI_MODE_RDONLY for Reading.

I’ve assigned 1MB(long N=5152*512/2) and calculated the buffer size to read data based on displacement from a file as follows.

```
bufsize= N/size;
nints= bufsize/sizeof(long);
long buf1[nints];
```

Then,I've used MPI_File_read_at to read data as int MPI_File_read_at(fh, rank*bufsize, buf, nints, MPI_LONG, &status1);And Close the file using: MPI_File_close(MPI_File fh). Moreover, I added error handling to all the MPI Functions. If error occurs, it will throw an error. And deleted temporary file.

And using MPI_Wtime(),I’ve calculated the reading, writing timings for each rank.