# Assignment - A2

Roll No. 202cd005

## * MPI - Theory *

GOODLUCK | Page No. 11
Date 28 / 11 / 20

1] In MPI, when does a non-blocking recv message return?

→ Non-blocking communications are done using MPI_Isend() & MPI_Irecv(). These function return immediately (i.e. they do not block) even if the communication is not finished yet.

2] What is an MPI communicator? What is MPI_COM_WORLD?

→ A MPI communicator is an object describing a group of processes.

MPI_COMM_WORLD → comprises all processes that are started by mpiexec or some related program.

MPI_COMM_SELF → contains only current process.

MPI_COMM_NULL → Invalid communicator. Routines that construct communicator can give this as result if error occurs.

3] In MPI, What is a process rank?

→ Process rank is an integer identifier assigned to each process within a communicator. It is unique for each process.

4] True or False: In MPI you set the number of processes when you write the source code. → False

5) Explain if the following MPI code segment is correct or not & why:

Process 0:

```
MPI_Recv(&yourdata, 1, MPI_FLOAT, 1, tag, MPI_COMM_WORLD, &status)
MPI_ISend(&mydata, 1, MPI_FLOAT, 1, tag, MPI_COMM_WORLD)
```

Process 1:

```
MPI_Recv(&yourdata, 1, MPI_FLOAT, 0, tag, MPI_COMM_WORLD, &status)
MPI_Send(&mydata, 1, MPI_FLOAT, 0, tag, MPI_COMM_WORLD)
```

→ (i) This segment of code is incorrect because it'll ~~to~~ end up in deadlock.
Here, as both of the processes are first ~~rec~~ calling receive function. Hence the ~~execution~~ Both of those will wait for completing receive block.

(ii) The correct order should be as follows:

Process 0:

```
MPI_Send(....);
MPI_Recv(....);
```

Process 1:

```
MPI_Recv(....);
MPI_Send(....);
```

6] Suppose that process 0 has variable A & process 1 ^also has a variable A. Write MPI pseudo-code to exchange these 2-values betn the processes.

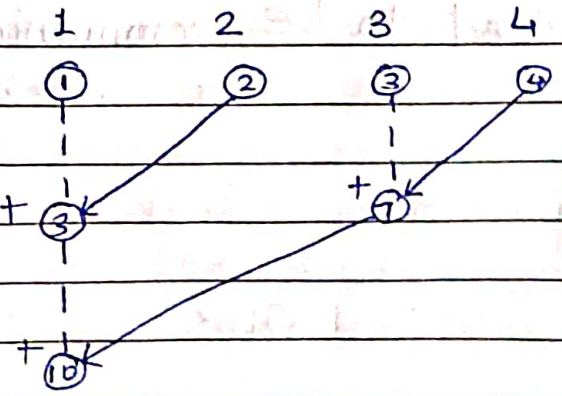
→ Pseudocode:-

```
if (myrank==0)
 { A=10
   MPI_Send(&A, 1, MPI_datatype, 1, 0, MPI_COMM_WORLD)
   MPI_Recv(&A, 1, MPI_datatype, 1, 1, MPI_COMM_WORLD, &status)
 }
else if(myrank==1)

{ A=20
  MPI_Recv(&A, 1, MPI_datatype, 0, 0, MPI_COMM_WORLD
                                      , &status)
  MPI_Send(&A, 1, MPI_datatype, 1, 1, MPI_COMM_
                                      WORLD)
}
```

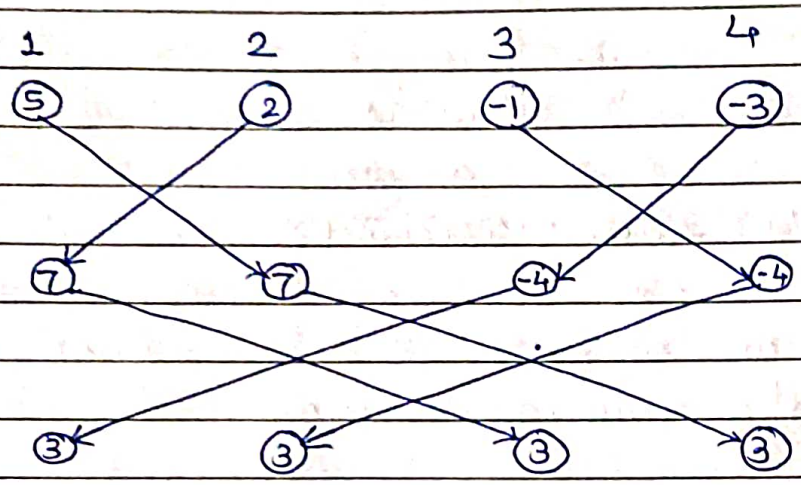
7] Explain Tree structured communication & Butterfly communication in MPI-implementation.

→ (a) Tree-structured communication.



(i) In this approach, the work of computing the operation (in this case sum) is distributed to some of the participating processes.
(ii) In this example 1 & 3 calculate intermediate result & communicate them to next process.
(iii) This results in most of the calculation being complemeted concurrentely.
(iv) Here, root node is not responsible for all calculations.

(b) Butterfly structured communication:




(i) This structure can be ~~ex~~ implemented using a similar scheme as that of tree structure.
(ii) The pairings are created using exclusive OR.
(iii) Terminating condition is different as the partial results need to be communicated multiple times.

8] Explain purpose of each library calls listed.

① MPI-Init
→ It is used to initiate an MPI computation.

② MPI-finalize → Terminate a computation

③ MPI-Comm_rank → MPI_COMM_RANK (comm, pid)
Determine identifies of current process.
IN comm communicator (handle)
OUT pid process id in group of comm (integer)

④ MPI-COMM-SIZE(comm, size)

(i) Determine the no. of processes in computation.
(ii) IN comm communicator (handle)
OUT size number of processes in group of comm (integer)

⑤ MPI-Send (buf, count, datatype, dest, tag, comm)

(i) Send a message.
(ii) IN buf address of send buffer (e.g. &yrdata
IN count number of elements to send (int >0)
IN datatype datatype of send buffer element
IN dest process id of destination process (int)
IN tag message tag (integer)
IN comm communicator (handle)

⑥ MPI_Recv (buf, count, datatype, source, tag, comm, stat)

(i) Receive a message
(ii) OUT buf address of receive buffer
IN count size of receive buffer, in elements (int)
IN datatype datatype of receive buff. elements.
IN ~~tag~~source process id of source or MPI_ANY_SOURCE (integer)
IN ~~tag~~ message tag or MPI_ANY_TAG (int)
IN comm communicator.
OUT stat status object.

⑦ MPI_Barrier (comm)

(i) Process synchronization (blocking)
(ii) All processes are forced to wait for eachother
(iii) Will reduce parallelism.
(iv) IN comm communicator.

⑧ MPI_Bcast (buf, count, datatype, root, comm)

(i) Broadcasts a message from the process with rank "root" to all other processes of the communicator.
(ii) IN/OUT buf address of buffer. (choice)
IN count number of entries in buffer (int)
IN datatype data type of buffer (handle)
IN root rank of broadcast root (int)
IN comm communicator (handle)

(9) MPI-Scatter (& sendbuf, sendcnt, sendtype, & recbuf, recvcnt, recvtype, root, comm)

(i) sends individual messages from the root process to all other processes.

(ii) IN sendbuf address of send buffer (choice, significant only at root)
IN sendcnt No. of elements sent to each process (int, significant only at root)
IN sendtype datatype of send buffer elements. (significant only at root) (handle)
OUT recvbuff address of recieve buffer (choice)
IN recvcnt no. of elements in receive buffer (integer)
IN recvtype data type of recieve buffer elements (handle)
IN root Rank of sending process (int)
IN comm communicator (handle).

(10) MPI-Gather (& sendbuf, sendcnt, sendtype, & Recv buf, recv cnt, recvtype, root, comm)

(i) Gathers together values from a group of processes.

(ii) All parameters are same as MPI-scatter, except here,
IN recvcnt number of elements for any single receive (int, significant only at root)

IN root rank of receiving process (int)

(ii) MPI-Reduce (&sendbuf, &recvbuf, count, datatype, op, root, comm)

(i) Reduces values on all processes to single value.

(ii)

| | | |
|---|---|---|
| IN | sendbuf | address of send buffer (choice) |
| OUT | recvbuf | address of recv. buffer. |
| IN | count | No. of elements in send buf (int) |
| IN | datatype | data type of elements in send buffer (handle) |
| IN | op | reduce operation (handle) |
| IN | rank | rank of root process (int) |
| IN | comm | communicator (handle). |

9) What is an MPI derived datatype? & When do we use one? Give example.

→ (i) MPI allows programmer to create their own data type, analogous to defining structures in C.

(ii) These are called derived datatypes.

(iii) MPI data types are mostly of use if you want to send multiple items in one message

(iv) Derived types allows to create new configuration contiguous data type consisting of an array of elements of another datatype.

(v) A vector data type consisting of regularly spaced blocks of elements of a component type; not regularly spaced data, there is the indexed data type, where you specify an array of index locations for block of elements of a component type.

(vi) struct data type can accomodate multiple data types.

(vii) A general datatype is an opaque object that specifies 2- things:
(a) A sequence of basic data types.
(b) A sequence of integer (byte) displacements.

Note:- Displacements are not required to be +ve, distinct or in increasing order.

(viii) Process to create and use a new datatype.

(a) Create your own data type using Datatype Constructors.

(b) Commit your datatype.

(c) Use it in communications routines.

(d) Free your data type.

* When to use? & example,
Consider, if you want to send a message that contain
(1) non-contiguous data of a single type.

② Contiguous data of mixed type
③ Non-contiguous data of mixed type.

Here we cannot use standard MPI-data types to send our message.
Hence we have to create our own data type i.e a derived data type.
So in short,
In cases where we cannot use standard MPI data types to send our message, we create a derived data type.

10] Explain idea of Parallel sort algorithm & its implementation in MPI.

→ (i) If we have a total of 'n' keys & p(= comm_sz) processes, our algorithm will start & finish with 'n/p' keys assigned to each process (Assuming n is evenly divisible by p).

(ii) At start there are no restrictions on which keys are assigned to which prosses. However, when the algorithm terminates,

(a) the keys assigned to each process should be sorted in (say) increasing order.
(b) if $0 \le q \le r < p$, then each key assigned to process q should be less than or equal to every key assigned to process r.

(iii) So, if we lined up keys according to

process rank - keys from process 0 first, then from process 1 & so on, then the keys would be sorted in increasing order.

11] When does a blocking send return?

→ Blocking communications are done using MPI_Send() & MPI_Recv(). These functions donot return until the communication is finished.
Hence, a blocking send i.e. MPI_Send() returns only after a communication is finished.