**Part 1**

MPI exercises

# Communication Problems

Given the following pseudo-code:

MPI\_Comm\_rank (comm, &rank); if (rank == 0) {

MPI\_Send (sendbuf, count, MPI\_INT, 1, ...); MPI\_Recv (recvbuf, count, MPI\_INT, 1, ...);

} else if (rank == 1) {

MPI\_Send (sendbuf, count, MPI\_INT, 0, ...); MPI\_Recv (recvbuf, count, MPI\_INT, 0, ...);

}

* 1. Indicate if some problem can arise when executed on two different processors.

As both processes send at the same time, they can block the subsequent process which is the MPI\_Recv leading to a situation of **deadlock.** Since both processes are waiting for each other to receive the message they sent, none of them will progress, resulting in a deadlock.

* 1. In case of finding a potential problem, write a new pseudo-code that solves the problem.

MPI\_Comm\_rank (comm, &rank);

if (rank == 0) {

MPI\_Send (sendbuf, count, MPI\_INT, 1, ...);

MPI\_Recv (recvbuf, count, MPI\_INT, 1, ...);

} else if (rank == 1) {

MPI\_Recv (recvbuf, count, MPI\_INT, 0, ...);

MPI\_Send (sendbuf, count, MPI\_INT, 0, ...);

}

# Collective Communication

Given the following excerpt of MPI parallel code which sends a matrix named B from one processor to the rest, answer the questions below.

...

/\* Replicate matrix B of size n\*n in each process \*/ if (!mpiRank) {

for (i=1; i<mpiSize; i++) {

MPI\_Send(B, n\*n, MPI\_DOUBLE, i, TAG\_INIT, MPI\_COMM\_WORLD);

}

}

else { /\* Receive B in each other process \*/

MPI\_Recv(B, n\*n, MPI\_DOUBLE, 0, TAG\_INIT, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

}

...

* 1. Which is the *rank* of the process that sends matrix B to all the processes?

Rank 0 in the !mpiRank

* 1. Write a single statement using an MPI collective communication primitive that performs the same work.

MPI\_Bcast(B, n\*n, MPI\_DOUBLE, 0, MPI\_COMM\_WORLD);

# Creating a new collective primitive

Assume we are designing a new collective routine for MPI. The syntax and semantics are as follows:

void MPI\_Round( void \*SendBuffer, /\* in \*/

void \*RecvBuffer, /\* out \*/ int count, /\* in \*/ MPI\_Datatype datatype, /\* in \*/ int tag, /\* in \*/ MPI\_Comm comm, /\* in \*/ MPI\_Status \*status); /\* out \*/

Semantics: The process that calls this funtion with logic identifier taskid within communicator comm, sends SendBuffer to process taskid-1 and receives RecvBuffer from process taskid+1. Note that the process with taskid 0 will send to the last process. And the last proces, will receive from process 0. Communications for both sending and receiving are both done with tag tag, and the status about the message reception in RecvBuffer is returned through status. The Buffers have count elements of type datatype. This routine assumes that buffers SendBuffer and RecvBuffer are not overlapped in memory. An example of utilization follows:

#include <mpi.h> int rank;

int nproc;

int main( int argc, char\* argv[] ) { MPI\_Status status;

MPI\_Init( &argc, &argv );

MPI\_Comm\_size( MPI\_COMM\_WORLD, &nproc ); MPI\_Comm\_rank( MPI\_COMM\_WORLD, &rank ); int sendBuffer[N];

int recvBuffer[N];

/\* Some useful work here \*/

...

/\* Being a collective communication it is called by all processes. \*/ MPI\_Round(sendBuffer,recvBuffer,N,MPI\_INT,1,MPI\_COMM\_WORLD,&status);

...

MPI\_Finalize();

}

* + 1. Provide an implementation of routine MPI Round which does not cause deadlocks in the communication.

Hint: Inspect and understand the code in ringbl/ringbl.c.soln or ringnb/ringnb.c.soln

within lab1/MPI/Tutorial.