ICHEC
Irish Centre for High-End Computing

ACM40660/PH504 Practical 6

ICHEC

2022/23 Spring

1 Hello world!

Build the MPI hello world example from Fig. 1.

```c
#include <stdio.h>
#include <mpi.h>

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

  int ierror;
  int myRank,uniSize;
  int version, subversion;
  int iMyName;
  char myName[MPI_MAX_PROCESSOR_NAME];

  ierror=MPI_Init(&argc,&argv);
  ierror=MPI_Comm_size(MPI_COMM_WORLD,&uniSize);
  ierror=MPI_Comm_rank(MPI_COMM_WORLD,&myRank);
  ierror=MPI_Get_processor_name(myName,&iMyName);
  ierror=MPI_Get_version(&version,&subversion);
  printf("I am process %d out of %d running on %s with MPI version %d.%d\n",
      myRank,uniSize,myName,version,subversion);
  ierror=MPI_Finalize();
  return ierror;
}
```

```fortran
program helloMPI
  use mpi
  implicit none

  integer :: ierror
  integer :: myRank,uniSize,version,subversion
  character(len=MPI_MAX_PROCESSOR_NAME) :: myName
  integer :: status(MPI_STATUS_SIZE),iMyName

  call MPI_Init(ierror)
  call MPI_Comm_size(MPI_COMM_WORLD,uniSize,ierror)
  call MPI_Comm_rank(MPI_COMM_WORLD,myRank, ierror)
  call MPI_Get_processor_name(myName,iMyName, ierror)
  call MPI_Get_version(version,subversion, ierror)
  write(*,'(a,i0,a,i0,a,a,a,i0,a,i0)')"I am process ", myRank, &
              " out of ", uniSize, " running on ", trim(myName), &
              " with MPI version ", version,".",subversion
  call MPI_Finalize(ierror)
end program helloMPI
```

Figure 1. MPI hello world, top C and bottom Fortran

1. Run it using one process and using 12 and 24 processes. Use (mpirun -n # ./prog) to run a MPI enabled program. The (-n) option takes a number, which is the number of MPI processes.

2. Change the code so it prints a message before MPI_Init call. When you run it using 12 processes, how many time is this message printed.

3. Change the code so that only one process handles the output. Choose yourself the right process for that.

2 MPI_Send()/MPI_Recv()

Write a simple program that performs communication between two processes P_0 and P_1. The alogirthm is as follows:

1. P_0 sends the string "Hello World! to P_1 using blocking MPI_Send().

2. P_1 receives this message using blocking MPI_Recv() and print.

   Try using MPI_Send() and MPI_Recv() following the rules given in Figure 2. We will cover MPI Point to Point Communication next week.

```
int MPI_Send(const void *buf, int count, MPI_Datatype datatype, int dest,
             int tag, MPI_Comm comm)

int MPI_Recv(void *buf, int count, MPI_Datatype datatype, int source,
             int tag, MPI_Comm comm, MPI_Status *status)
```

```
MPI_SEND(BUF, COUNT, DATATYPE, DEST, TAG, COMM, IERROR)
        <type>    BUF(*)
        INTEGER    COUNT, DATATYPE, DEST, TAG, COMM, IERROR

MPI_RECV(BUF, COUNT, DATATYPE, SOURCE, TAG, COMM, STATUS, IERROR)
        <type>    BUF(*)
        INTEGER    COUNT, DATATYPE, SOURCE, TAG, COMM
        INTEGER    STATUS(MPI_STATUS_SIZE), IERROR
```

Figure 2. MPI_Send() and MPI_Recv() syntax, top C and bottom Fortran

Input Parameters:

- buf: Initial address of send/receive buffer.
- count: Number of elements sent/received.
- datatype: MPI Datatype of each send/receive buffer element.
- dest: Process id of destination/source.
- tag: Message tag (User defined integer).
- comm: MPI Communicator.

Output Parameters:

- status: Status object.
- IERROR (Fortran only): Error status.

```
MPI_CHAR - char
MPI_SHORT - short
MPI_INT - int
MPI_UNSIGNED - unsigned int
MPI_LONG - long
MPI_FLOAT - float
MPI_DOUBLE - double
```

```
MPI_REAL - REAL
MPI_INTEGER - INTEGER
MPI_LOGICAL - LOGICAL
MPI_DOUBLE_PRECISION - DOUBLE PRECISION
MPI_COMPLEX - COMPLEX
MPI_CHARACTER - CHARACTER
```

Figure 3. Main MPI Datatypes, top C and bottom Fortran