ICHEC
Irish Centre for High-End Computing

# Practical3

**ICHEC**

**April 26, 2020**

# 1 Hello world!

Build the MPI hello world example from Fig. 1.

1. Run it using one process and using 12 and 24 processes. Can you rely on the output order? 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.

4. Can you make the original code to print the messages in a deterministic order (ordered by rank)?

```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

## 2 Ping Pong

Write a simple ping pong program. This involves passing of a message between two processes $P_0$ and $P_1$. The algorithm is as follows

1. let us assume we have an initial message which contains the integer value 10.

2. $P_0$ increments this message by one and passes it to $P_1$ (ping)

3. $P_1$ receives the message it increments it by one and passes it back to $P_0$
4. the last two steps are repeated n times. (see fig. 2)

The code shall do

1. rank 0 prints the value of the message it has after n exchanges.
2. rank 0 prints the average time per exchange. **Hint:** use MPI_Wtime() function to get the time.
3. determine the value n for which the measured time is meaningful. **Hint:** check the resolution of the timer with MPI_Wtick()
4. use MPI_Send and MPI_Recv to pass the messages around.

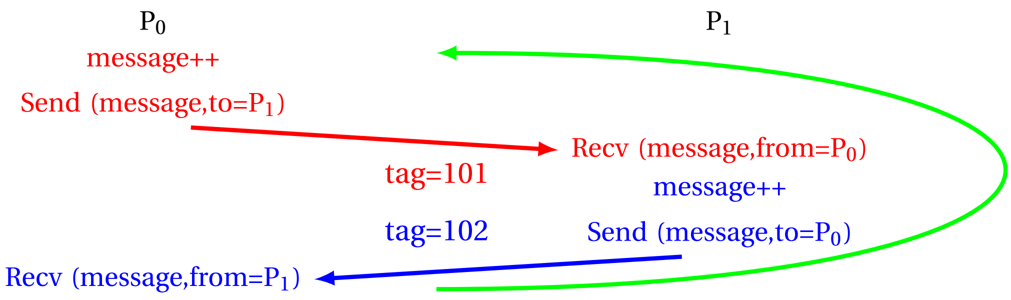

Figure 2. Ping Pong algorithm

## 3 Latency and Bandwidth

Latency is defined as the time to transfer a zero length message. Bandwidth is defined as the size of the message in bytes/ transfer time.

1. Modify the ping pong code to measure the latency (use MPI_BYTE as transfer type).
2. Modify the ping pong code to measure the Bandwidth (use double or real(kind=8)). Measure the bandwidth for the following sizes 8 B, 512 B, 32 KiB, 2MiB these correspond to arrays of length 1, $2^6$, $2^{12}$ and $2^{18}$, respectively.
3. do the measurements for the following two cases: i) inside one node (mpirun -ppn 2), ii) in between two nodes (mpirun -ppn 1)