
# PRACE Course: Intermediate MPI

## Day 1

**9/11/2022**

# 1 Ping Pong Benchmark

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. 1)

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. test with the following send modes: MPI_Send and MPI_Ssend.

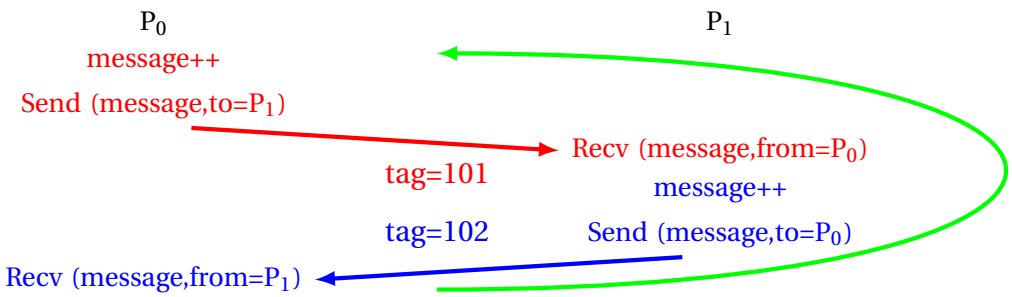


Figure 1. Ping Pong algorithm

5. Latency is defined as the time to transfer a zero length message. Modify the ping pong code to measure the latency (use MPI_CHAR as transfer type).
6. Bandwidth is defined as the siz e of the message in bytes/ transfer time. 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.

## 2 Communication in a Ring

1. Write a MPI program where each rank sends a message to its right neighbour.
2. The message is passed around the ring until it reaches the originator rank.
3. At this point the **message** should contain the sum of all the ranks.
4. So at termination all ranks should have the sum of the ranks.
5. Use non-blocking communications.
6. Write an equivalent operation that does the same using MPI_Ireduce().

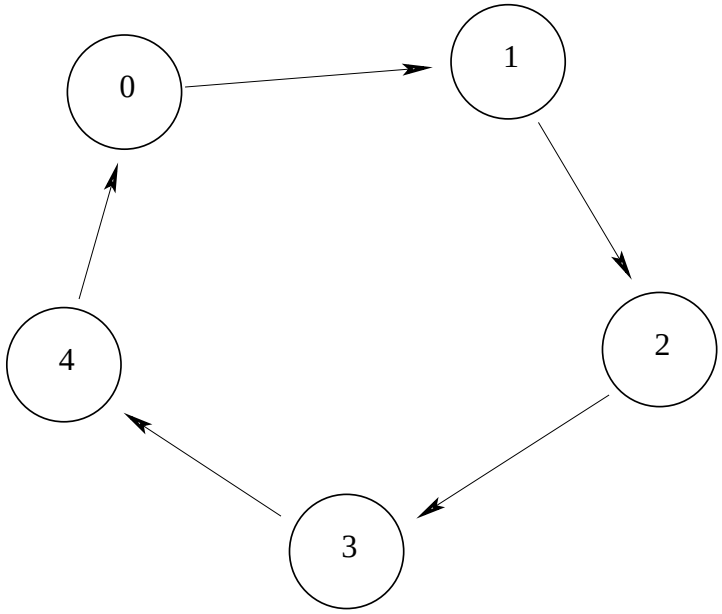


Figure 2. Five processes logically arranged in a ring: the left neighbor of $P_0$ is $P_4$ and its right neighbor is $P_1$.

## 3 Collectives: Allgather/Gatherv/Reduce

The serial code is given (inc_serial.c / inc_serial.f90). It increments every element of the array by two. Parallelise using MPI as follows.

1. A real vector *a* of size length is allocated at Process 0.
2. Each process contains a chunk of the vector a_per_process of size mylen_per_process. Length of *a* doesn't have to be a multiple of the number of processes. Calculate the

size of the subvector at each process, allocate the subvector and initialise as 0.0. Ex: length=10, uniSize=4, mylen_per_process at Process0 is 3, Process1 is 3, Process2 is 2 and Process3 is 2.

3. Each process gets the length of a_per_process from all other processes using MPI_Allgather, calculates the recvcounts and displs arrays.
4. Each process increments every element of the subvectors by 2.0 in parallel. Execute the increment operation n number of times and use MPI_Wtime() function to get the time.
5. Gather subvectors into a in Process 0 using MPI_Gatherv.
6. Each process has its own timing. The root process collects the times and output the min, max and average values of the times.