Parallel and High Performance Computing

Dr. Pablo Antolín

Solution Series 4

March 20 2025

MPI

Different parallelizations of the π reduction

If you pull from the git exercises repository, the solutions can be found in the `lecture_04/pi/solutions` folder.

1 First steps with MPI

Exercise 1.1: ***MPI: Hello, World!***

- For the initialization you can either uncomment the parameters `argc` and `argv`, and pass them to `MPI_Init`, or, according to the documentation in C, you can also simply do `MPI_Init(NULL, NULL)`.
- To compile, in `Makefile` `CXX` and `CC` have to be changed from `g++` to `mpicxx` (or to `mpiicpc` for using intel MPI compiler). To know what `mpicxx` does you can test `mpicxx -show` (or `mpiicpc -show`).

2 Using a Ring to communicate

Exercise 2.1: π ***MPI Ring (point to point synchronous)***

- Split the integral calculation among the processes by partitioning the indices of the `for` loop according to the rank of every process.
- As explained in the course, `MPI_Ssend` is blocked as long as the corresponding receive is not posted. In this case if you write `MPI_Ssend` followed by `MPI_Recv`, all the processes will stay stuck on `MPI_Ssend` (deadlock). That is why you should ensure that at least one process is doing a `MPI_Recv` before the `MPI_Ssend`.

  In the solution, all the even process ranks are doing send and then receive, while all the odd ranks do receive followed by send.

If you use `MPI_Send` instead of `MPI_Ssend` in this example, it will not deadlock since `MPI_Send` acts as `MPI_Bsend` on small buffers, and as `MPI_Ssend` for large ones[1]. Small and large are implementation dependent.

Exercise 2.2: *π MPI Ring (point to point synchronous sendrecv)*

With `MPI_Sendrecv` there is no need to do a conditional choice based on the rank.

Exercise 2.3: *π MPI Ring (point to point asynchronous)*

With `MPI_Isend` you can overlap the computation of the sum. But you have to be careful to wait for the end of the communication before changing the `send` buffer.
It is not because you already received data from the previous process that it means the send to the next process is already finished.

3 MPI collectives

Exercise 3.1: *π MPI (collective gather)*

In `MPI_Gather` you have to be careful with the arguments: The send buffer is of size 1, while the receive buffer on the root process, here the process 0, is a buffer of size (at least) `psize`, even though the argument specifying the receive size is 1. It is the size per process and not the size of the actual buffer.

Exercise 3.2: *π MPI (collective reduce)*

The exercise asks to use `MPI_Reduce`. This would get the sum stored on the process 0. If you want the solution on every process, you can use `MPI_Allreduce`, that is equivalent to a reduce followed by a broadcast.
In the solution we also use `MPI_IN_PLACE` as the send buffer, this makes the receive buffer act as a send and receive buffer. It can also by used with `MPI_Reduce`.

[1] Check the documentation in https://rookiehpc.github.io/mpi/docs/mpi_send/index.html.