MPI Practicals

In each chapter, there are exercises to complement MPI topics we will learn in the Summer School.

Working through the exercises is essential, and it will give you a better understanding and reinforce the concepts taught.

To compile your parallel codes, use CC for C++, cc for C and ftn for Fortran

To run your code:

- Allocate an interactive session:
- >salloc –res = course –N<*number of nodes*> (time for 1 hr)
- >aprun–n <*number for processes*> <yourcode>

Remember that a node on Todi has 16 cores (max 16 mpi processes per node)

Happy coding!

Hello MPI World!!

In folder 1.hello_mpiworld:

Exercise 1: hello_world

- Write a minimal MPI program which prints "hello world" by each MPI process
- Compile and run it on a single processor
- Run it on several processors in parallel

Exercise 2: hello_world_advanced

- Modify your program so that:
  - Each process write its rank and the size of MPI_COMM_WORLD
  - Only process ranked 0 in MPI_COMM_WORLD prints "hello world"
  - Is the output deterministic?

Point-to-Point Communication

In folder 2.point-to-point

Exercise 1: mpi_send_recv

- Send a number from one process to another
- The sending process gets the number, sends it to the receiver, and the receiver prints it out.
- What would happen if you interchanged the send/receive calls?

Exercise 2: ping_pong

Write the following program:

- Process 0 sends a message to process 1 (ping)
- After process 1 receives this message, it sends a message to process 0 (pong)
- Repeat this ping pong with a loop of length 50.
- There are already timing calls before and after the loops. After exiting the loop the code calculates and prints the time it takes to send one message.
- **Note down the number!**

Exercise 3: ping_pong_advanced1

Modify your ping-pong code such that you exchange one message before entering the 50 message loop.

- That excludes timing MPI startup "costs"
- What is the time of exchange of a message now?

Exercise 3: ping_pong_advanced2_send

Optional: Benchmarking MPI_Send.

Modify your ping_pong_advance1 code to:

- Print out the latency (transfer time) and, bandwidth (message size (bytes) / transfer time) for the following message sizes:
  8 bytes (only double)
  - 512 B (=8 x 64 bytes)
  - 32 kB (=8 x 64^2 bytes)
  - 2MB (= 8 x 64^3 bytes)

Replace send with ssend (synchronous send).

Non-blocking communication

In folder 3.non_blocking

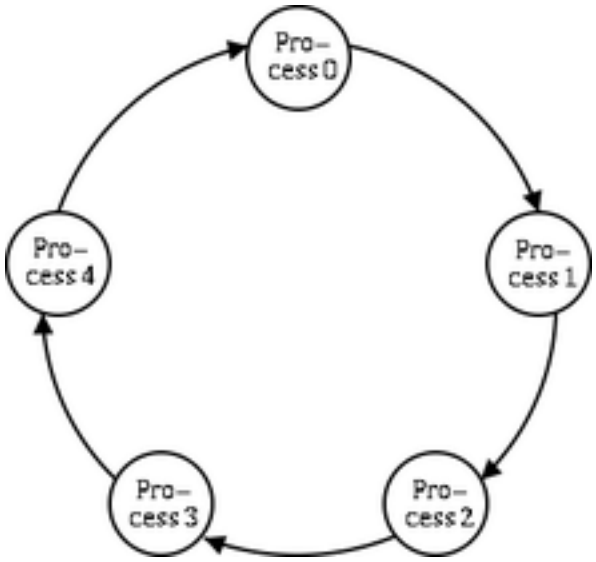
Exercise 1: ring (aka the tedious way to allreduce)

A set of processes are arranged in a ring.

The objective is for each process to receive a number from all the other processes, and add all the numbers together. To achieve this:

- Each process stores its rank into an integer variable snd_buf
- Each process passes this on to its neighbour on the right (so it receives a value from its left neighbour)
- Each processor calculates the sum of all values it receives
- Repeat steps above until each "message" goes once around the circle (size(MPI_COMM_WORD) iteration) to make sure all processes have received all numbers.

Hints: Use non blocking operations to avoid deadlocks.
Either Issend-Recv-Wait, or Irecv-send-Wait.
Just for fun, use blocking and see if it deadlocks!
Use modulo arithmetic to calculate the ranks of left and right neighbors
In C with % operator, Fortran mod()
Fortran users: users: After calling MPI_Wait, call mpi_get_address(snd_buf, dummy), where dummy is an integer (KIND=mpi_get_address)



Exercise 2: C: ghost_cell_ex_mpi_row F: ghost_cell_mpi_column
This exercise starts looking into 2D domain decomposition.

You have a 40 x 40 grid of data, and want to distribute the work to 4 x 4 processes (so each process has 10 x 10 data). For any "real" problem though, each process might also need to communicate some of its data (on boundaries) to its neighbor.

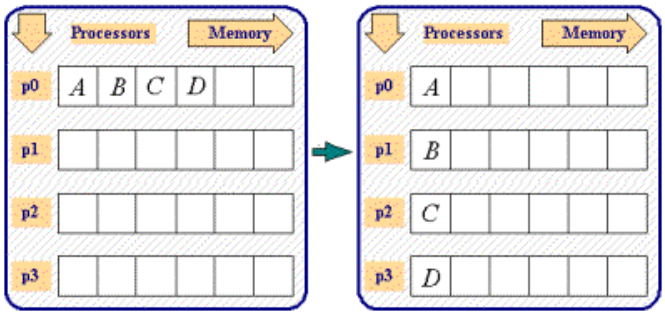
For C: communicate top/bottom row to top/bottom neighbor
For Fortran: Communicate left/right column to left/right neighbor

Using (either):

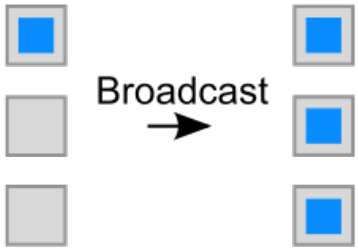
A) MPI_Irecv, MPI_Send, MPI_Wait
B) MPI_Send, MPI_Irecv, MPI_Wait
C) MPI_Sendrecv

The code will print out the data and ghost cells of your rank of choice.

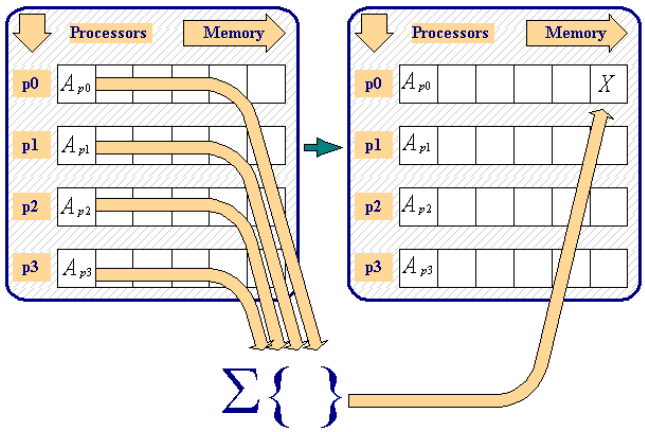
Exercise 1: scatter_mpi
Here process 0 receives an array of data (either from terminal, or you can initialize it yourselves) and scatters the data to the other processes. (in this case, one element to each process).



Exercise 2: broadcast_mpi
Broadcast the value of process 0 to all other processes.



Exercise 3: reduce_mpi
Use mpi_reduce to sum the values help by each process. The result will be stored by rank 0
Use mpi_allreduce to sum the valued held by each process. The result will be stored by all ranks



Exercise 4: ring_allreduce
Remember the ring exercise from the Point-to-Point chapter? Simplify the initial code using allreduce!

Derived Datatypes

In folder 5.derived_datatypes

Exercise: ghost_cell_ex_datatype_
This is the continuation of exercise 2 from chapter3 (non-blocking). In that exercises you exchanged top/bottom ghost cells (C) , or left-right (Fortran). To complete communications, you need to exchange the left/right (C) and top/bottom(F). Since the column data in C (row data in Fortran) are not contiguous in memory, as in the previous exercise, you need to create a datatype to store the columns (rows) to be exchanged.

The solution from chapter 3 is provided. You will need to:

1) Find out who your neighbours are!
2) Create a vector derived datatype to store the C-Column (F-row) with the ghost cell data
3) Communicate those to the appropriate neighbours as before.

Make sure the results make sense! (print data from different ranks)

Virtual Topologies

Exercise 1: topology_ring
This builds on our original version of the ring exercise, without the allreduce. You will use a Cartesian communicator to find your neighbors

- Set up a 1-D Cartesian communicator
- Use MPI_Cart_shift to find your left and right neighbours
- Use the new communicator to complete communication

Exercise 2: ghost_cell_cart_ex_mpi, revisited
You have already seen this example in non-blocking, and derived datatypes. Now we will simplify, using a Cartesian grid communicator!

- Create a new 2D Cartesian communicator from MPI_COMM_WORLD.
- Use the new communicator, and MPI_Cart_shift to calculate the ranks of your neighbors in all directions. (i.e. no more modular arithmetic!!)

At first, we will not use the new communicator for communication, so you can compare the results with the old code. Once you are satisfied, you can use rank reordering (reorder = 1), and use the new_comm for communications. Compare the results with the previous ones (they could be the same!)