Computação em Larga Escala

(ano letivo 2024'25)

3 - MPI Exercises

The Message Passing Interface (MPI) is a standardized and portable API used for parallel computing on distributed memory systems. MPI allows processes to communicate with one another by sending and receiving messages, making it the foundation for many large-scale scientific applications.

In this set of exercises, you will practice writing MPI programs to develop an understanding of point-to-point communication, collective operations, and distributed data handling.

🔥 **Recommended Setup**

It is highly recommended to install the **MPICH** implementation of MPI on your machine. You can usually install it using your system package manager (e.g., `apt`, `brew`, `yum`), or download it from https://www.mpich.org/downloads/.

3.1 - Ring Communication

Modify the provided `sendRecvData.c` program to circulate the message `"I am here (<process rank>)!"` in a ring topology across all processes. Each process should receive the message, append its own information, and send it to the next process.

Hint: Use the following MPI functions:

- `MPI_Comm_rank`
- `MPI_Comm_size`
- `MPI_Send`
- `MPI_Recv`

Also consider how to handle wrap-around from the last to the first process to form a ring.

3.2 - Alive and Well

Write a program where process 0 sends a message saying it is "alive and well" to all other processes. Each receiving process replies with a similar message. Process 0 should then print all responses.

Hint: Use a combination of:

- `MPI_Send`
- `MPI_Recv`
- `MPI_Comm_size`
- `MPI_Comm_rank`

Consider how to loop over `MPI_Recv` calls to gather all messages on process 0.

3.3 - Global Min and Max

Generate a sequence of random numbers on each process. Write a program that finds the global minimum and maximum values among all generated numbers.

Hint: Use collective communication:

- `MPI_Reduce` with `MPI_MIN` and `MPI_MAX`
- Or `MPI_Allreduce` if all processes should get the result

3.4 - Parallel Matrix Product

Write a program that multiplies two square matrices using multiple processes. Each process should compute a subset of the output matrix.

Hint:

- Use `MPI_Scatter` to distribute rows of the first matrix
- Broadcast the second matrix using `MPI_Bcast`
- Use `MPI_Gather` to collect the result

This will require careful setup of data distribution and indexing.