Master in High Performance Computing
Subject: Parallel Programming

Unit 3: Programming using MPI

The MPI labs will be performed ONLY in the Finis Terrae (ft3) supercomputer.

It is necessary to upload to AulaCesga only one zip file for the specific exercises including an explanation for each question (a md ot txt file), your source code and the scripts. The first line of this file must include your surname and name.

Starting MPI codes:

First of all, it is necessary to open a xterm in ft3, using ssh or similar command, and to load the following modules:

```
$ module load intel impi
```

LAB 1: Compile and run a MPI example using the queue system

It is necessary to save the files: example1_hello_rank_size.c and job_ft3.sh in the same directory.

1. Check the code
2. Login in FT3, using ssh or similar, but do not use ***compute*** command
3. Compile the MPI code using ***mpicc*** in the interactive session. If you use "math.h" library, it is necessary to include -lm option. This command creates an executable file called *example1*
   ```
   $ mpicc  -o  example1 example1_hello_rank_size.c -lm
   ```

   If the opcion -o is not used, then the executable file is called a.out
4. Send the job to the queue system with ***sbatch*** command.
   ```
   $ sbatch  job_ft3.sh
   ```
5. You can use ***squeue*** command to check the queue system.
6. It is possible to use a different number of tasks using option -n. NOTE: default queue allows only one node.
   ```
   sbatch [-n ntasks] [-t time] ….
   ```

   Example using 6 MPI tasks and only one node, 5 minutes.
   ```
   $ sbatch -n 6  -t 05:00 job_ft3.sh
   ```
   (options in sbatch commands have priority over file.sh ones)

7. If you want to use a different nodes you can use option --ntasks-per-node.
   Example: you want to use 4 tasks with only 2 task per node -> you request 2 nodes.
   sbatch -n 4 --ntasks-per-node 2 -t 05:00 job_ft3.sh

   NOTE: It is highly recommendable to use only one node for testing and to reduce the requested time as much as possible!!!

Execute the code 4 times using 6 processors and answer these questions:

* Do you always obtain the same output?

* Is the output of the processes in some special order?

LAB 2: Send/receive

Download the file example2_send_receive.c and execute this code using 5 processes. Answer these questions:

* Do all the processes obtain the same output?

* Is the output of the processes in some special order?

* It is necessary to indicate the size of the message when you send and receive it. Try with different size values, smaller and larges than the correct one, and check what it is happening.

* If one of the processes has not call the MPI_Recv or MPI_Send what happens?

LAB 3: Sendrecv

There is another command to send/receive a message.

```
int MPI_Sendrecv(const void *sendbuf, int sendcount, MPI_Datatype sendtype, int
dest, int sendtag, void *recvbuf, int recvcount, MPI_Datatype recvtype, int
source, int recvtag, MPI_Comm comm, MPI_Status *status)
```

Download the file example3_sendrecv.c, and execute the code using 4 processes. Check the output and try to understand the code. Explain the behavior of the code.

LAB 4: Send/receive a vector

Download the file example4_send_receive_vector.c and execute the code using only 2 processes. In this case, process 0 sends a row of its matrix to process 1. Check the output and try to understand the code. Answer these questions:

* Check the status of the reception (*info* variable)

* Obtain the total communication time. Is it possible that communication time for process 1 will be higher than process 0? Explain your answer.

LAB 5: Send/receive deadlock

Download the file example5_send_receive_Deadlock.c, and execute this code using 2 and 4 processes. Answer these questions:

* Does the code run correctly?

* Implement several solutions using all possible send functions using synchronous reception. Make a table with the send and receive time for each function using message sizes: 10^2, 10^4 and 10^6 doubles.

LAB 6: Irecv

Download the file example6_receive.c, and execute this code using 2 and 4 processes. Change the MPI_Recv function to MPI_Irecv. Answer these questions:

* Implement a version of the code using MPI_Wait. Obtain the execution time used by MPI_Irecv and MPI_Wait and compare with the time original time using MPI_Recv

* Implement a new version of the code using MPI_Test. Obtain the execution time used by MPI_Irecv and MPI_Test and compare with the time original time using MPI_Recv

LAB 7: Broadcast

Download the file example7_Broadcast.c, and execute the code using 6 processes. Answer these questions:

* Which is the process that sends the message?

* Which are the processes that receives the message?

* Change the code, now the emitter of the message should be process 1. Obtain the communication time.

* Implement an alternative version of Broadcast using send/receive functions. Obtain the communication time for this case and compare the obtained value with the Broadcast function.

LAB 8: Scatter/Gather

Download the file example8_Scatter.c and execute the code using 4 processes. Answer these questions:

* Change the code to send two floats to each process

* Now each process must send its rank to 0 process using MPI_Gather. Change the code to use a generic number of processes

* Write a new version using MPI_Allgather based on the previous example. We want each process receives the same vector

LAB 9: Reduce

Download the file example9_Reduce.c and execute the code using 5 processes. Answer these questions:

* Run the code four times and make a table with these execution times. Do you obtain always the same or similar execution time?

* Implement a version to obtain the results in all the processes using MPI_Allreduce

* Sometimes it is necessary to include a MPI barrier to be sure all the processes are in the same function at the same time. Is it necessary in this case?

LAB 10: Scan

Download the file example10_Scan.c and execute the code using 4 processes. Answer these questions:

* Implement a new version to obtain the maximum and minimum local probabilities

* Which are the main differences with MPI_Reduce function?

LAB 11: Reduce_scatter

Download the file example11_Reduce_scatter.c and execute the code using 4 processes. Answer these questions:

* Explain how the function MPI_Reduce_scatter works in this example.

* Fill in the next table with the sendbuf data and check if the obtained results are correct

Process 0	Process 1	Process 2	Process 3	Result