Université Grenoble Alpes, Grenoble INP, UFR IM$^2$AG

Master 1 Informatique and Master 1 MOSIG

# UE Parallel Algorithms and Programming
Lab # 3

F. Desprez (Inria), J-F. Méhaut (UGA/Polytech), T. Ropars (UGA/IM$^2$AG), E. Saillard (Inria)

March 21st, 2017

Every MPI program:

- Includes the MPI header file `mpi.h`
- Calls a routine to initialize the MPI environment (`MPI_Init`)
- Calls a routine to terminate the MPI environment (`MPI_Finalize`)

Compilation and execution of a MPI program:

- Compilation: `mpicc -o myprogram myprogram.c`
- Execution with 4 MPI processes using Open MPI (available on the machines in the lab room): `mpirun -n 4 ./myprogram`

## Exercise 1: Hello World!

Write a `hello_world` program in which each process displays its ID (rank) and the total number of processes.

## Exercise 2

Modify your `hello_world` program so that processes with an even rank number print a different message from the other processes.

## Exercise 3

Until now, all MPI processes are running on the local host. In this exercise, we would like to execute MPI processes on several machines of the lab room. To do so, the first step is to allow ssh connections between hosts.

- Generate ssh keys without password

```
ssh-keygen -t rsa
```

- Add the new public key to the list of authorized keys

  ```
  cd ${HOME}/.ssh; cat id_rsa.pub >>authorized_keys;
  ```

- Prevent ssh from checking host key at first connection by adding the following lines to `${HOME}/.ssh/config`:

  ```
  Host *
     StrictHostKeyChecking no
  ```

Option `-hostfile myfile` should be used to specify to `mpirun` on which hosts to run the MPI processes. File `myfile` contains a list of host names.

Modify the `hello_world` program so that each process gets and displays the hostname of the machine it is running on. Try running this program on multiple hosts. Make the number of processes vary to observe how Open MPI places processes by default.

## Exercise 4: Simple communication

Write a program in which process with rank 0 sends an array of 10 64-bit floating point numbers to the process with rank 1. Process 0 will fill in the array and process 1 will print it.

## Exercise 5: ping/pong

Modify the previous program so process 1 sends the modified array back to process 0. Modify the program again to measure round-trip time using `MPI_Wtime()`.

## Exercise 6: Communication ring

Write a program in which a token is passed among processes in a ring. At the beginning, process 0 owns the token. Then it is passed from process to process (with modification) until reaching process 0 again.

## Exercise 7: Non-blocking communication

Write a program in which two processes want to send a message to each other at the same time (send and receive). Try a version using blocking communication primitives and then correct it using non-blocking communication primitives. Make tests with small and large messages to see whether the behavior changes.

# Bonus

## Exercise 8: Broadcast

Write a program in which process 0 sends an array of 10 integers to all other processes.

## Exercise 9: Total sum

Write a program which computes the sum of all processes ID and sends the result to all of them. Write two versions: one with a reduction followed by a broadcast and another one using a global reduction.

## Exercise 10: Barrier

Write a program in which processes synchronize. Use loops or `sleep` functions to obtain different execution time among processes.