# MPI Labs - Supplementary material

Álvaro Moure

April 29, 2025

# Loading modules in the cluster

Basic commands on how to use Environment Modules in the cluster:

```
$ module list
$ module purge
$ module load
$ module spider [module_name]
$ module unload
```

Click here to open Environment Modules Documentation

# Loading modules in the cluster

In this practice we will need to load a specific version of the GNU compiler and the OpenMPI software to access the MPI compiler, libraries and runtime:

```
$ module spider openmpi

----------------------------------------------
  openmpi: openmpi/5.0.3
----------------------------------------------
    Description:
      Open MPI is a powerful and widely-used ...

    You will need to load all module(s) on ...
      gcc/13.3.0
      gcc/8.5.0
```

# Loading modules in the cluster

In this practice we will need to load a specific version of the GNU compiler and the OpenMPI software to access the MPI compiler, libraries and runtime:

```
$ module load gcc/13.3.0
$ module load openmpi/5.0.3
$ module list

Currently Loaded Modules:
  1) gcc/13.3.0   2) openmpi/5.0.3
```

**This is needed every new login!**

We can do better...

# Loading modules in the cluster

A good practice when working with multiple modules, which can vary depending on the job to launch, is to load modules in your batch script (`job.sh`):

```
#!/bin/bash

#SBATCH --job-name=test
#SBATCH --output=job_%j.out
#SBATCH --error=job_%j.err
#SBATCH --partition=std
#SBATCH --ntasks=4
#SBATCH --time=00:00:05

module purge
module load gcc/13.3.0 openmpi/5.0.3

mpirun -np 4 <exec> [args] # or srun <exec> [args]
```

# Monte Carlo Exercise

What is asked to do?

1. Start the `montecarlo.c` source file with a MPI "`Hello World`" implementation.
2. Use the "`Hello World`" to start testing the Makefile and job submision in the pirineus3 cluster.

# Makefile

Makefile from Cholesky:

```
CC=gcc
CFLAGS=-O2 -fopenmp -march=native -lm -std=c99
OBJ=cholesky

all:
    $(CC) main.c $(OBJ).c -o $(OBJ) $(CFLAGS)

clean:
    rm $(OBJ)
```

# SBATCH directives

Reminder of the SBATCH directives to ask for more than one process:

```
Option 1 (simpler for what we will need)
#SBATCH --ntasks=number

Option 2 (more control over resource allocation)
#SBATCH --nodes=number
#SBATCH --tasks-per-node=number
```

# Monte Carlo Exercise

What is asked to do?

1. Start the `montecarlo.c` source file with a MPI "`Hello World`" implementation.
2. Use the "`Hello World`" to start testing the Makefile and job submision in the pirineus3 cluster.
3. Read the 3 command line arguments: `N, NUM_SAMPLES, SEED`
4. Distribute the number of points to compute among the number of processes.
5. Generate random $(x_1, x_2, \cdots, x_i)$ coordinates over the hypercubed domain. HINT: Number generator $x \in [0, 1]$, then in order to get $x \in [-1, 1] \rightarrow x = 2 * x - 1$.

# Monte Carlo Exercise

What is asked to do?

6. Count the number of points are inside the hypersphere: $\sqrt{x_1^2 + x_2^2 + \cdots + x_i^2} \leq 1$ for $i = 1, Dims$
7. After the counting, do you need any MPI communication? What about the timings?
8. Every MPI process has to time the Monte Carlo computation, and only the slowest must be printed. Again, any MPI communication needed?

# Flight Controller: Understanding the problem

Before starting to code we need to properly understand the problem and how it's coded:

1. Explore input data

```
# Plane Data
# Map: 100.00, 50.00 : 10 5
# Number of Planes: 5
# i x y vx vy
0 10.0 1.0 0.0 1.0
1 30.0 1.0 0.0 1.0
2 50.0 1.0 0.0 1.0
3 70.0 1.0 0.0 1.0
4 90.0 1.0 0.0 1.0
```

# Flight Controller: Understanding the problem

Before starting to code we need to properly understand the problem and how it's coded:

1. Explore input data
2. Create `Makefile` and `job.sh` for sequential program

```
CC=...
CFLAGS=-O3 -lm -std=c99
OBJ=...
OBJ2=...

seq:
    ...

par:
    ...

clean:
    rm $(OBJ) $(OBJ2)
```

# Flight Controller: Understanding the problem

Before starting to code we need to properly understand the problem and how it's coded:

1. Explore input data
2. Create `Makefile` and `job.sh` for sequential program
3. Learn how to run it

```
./fc_seq input_planes_test.txt ...
Flight controller simulation: ...
Time: comp: 0.00s    total: 0.00s
Ok! Plane 0 found at (10.00, ...)
Ok! Plane 1 found at (30.00, ...)
Ok! Plane 2 found at (50.00, ...)
Ok! Plane 3 found at (70.00, ...)
Ok! Plane 4 found at (90.00, ...)
```

# Flight Controller: Understanding the problem

Before starting to code we need to properly understand the problem and how it's coded:

1. Explore input data
2. Create `Makefile` and `job.sh` for sequential program
3. Learn how to run it
4. Navigate the code

```
./fc_seq input_planes_test.txt ...
Flight controller simulation: ...
Time: comp: 0.00s    total: 0.00s
Ok! Plane 0 found at (10.00, ...)
Ok! Plane 1 found at (30.00, ...)
Ok! Plane 2 found at (50.00, ...)
Ok! Plane 3 found at (70.00, ...)
Ok! Plane 4 found at (90.00, ...)
```

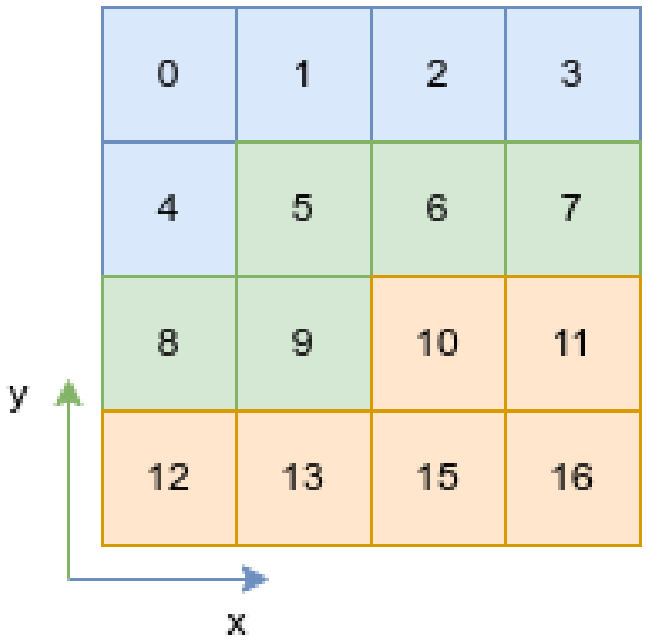
# Flight Controller: Understanding the problem

Before starting to code we need to properly understand the problem and how it's coded:

1. Explore input data
2. Create `Makefile` and `job.sh` for sequential program
3. Learn how to run it
4. Navigate the code
5. Navigate the MPI code, check TODOs

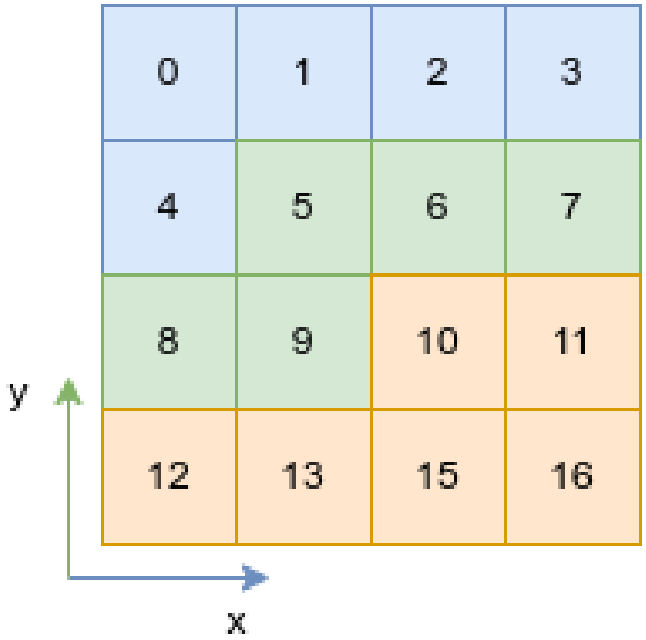
```
./fc_seq input_planes_test.txt ...
Flight controller simulation: ...
Time: comp: 0.00s    total: 0.00s
Ok! Plane 0 found at (10.00, ...)
Ok! Plane 1 found at (30.00, ...)
Ok! Plane 2 found at (50.00, ...)
Ok! Plane 3 found at (70.00, ...)
Ok! Plane 4 found at (90.00, ...)
```

# Flight Controller: Domain decomposition



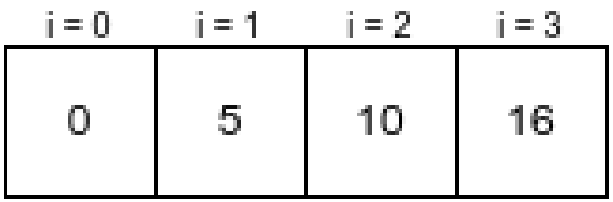

- Map Grid $N \times M$: $4 \times 4$
- MPI Comm Size $= 3$
- How do we decompose the domain?
- `total_displacements[size+1]`
- Each process start at index: $i * (gridCells)/size$
- And takes the following number of cells: `total_displacements[i+1] - total_displacements[i]`

# Flight Controller: Domain decomposition

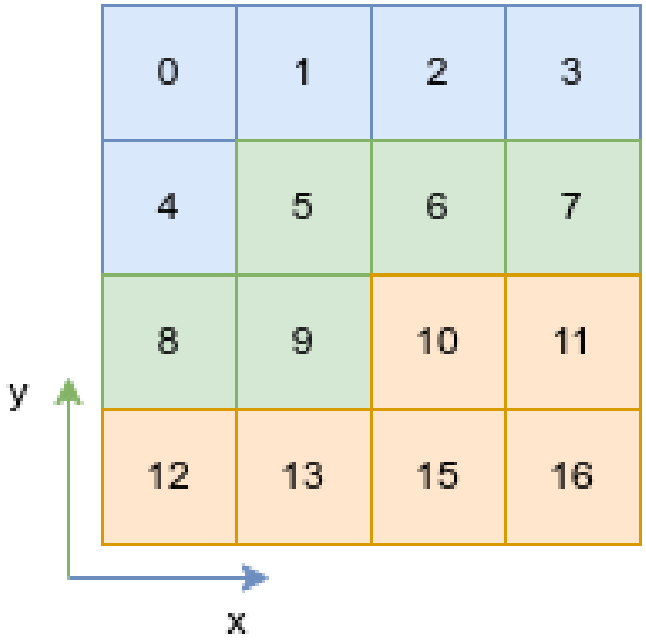



$i * (gridCells)/size$

- $i = 0 * 16/3 = 0$
- $i = 1 * 16/3 = 5$
- $i = 2 * 16/3 = 10$
- $i = 3 * 16/3 = 16$

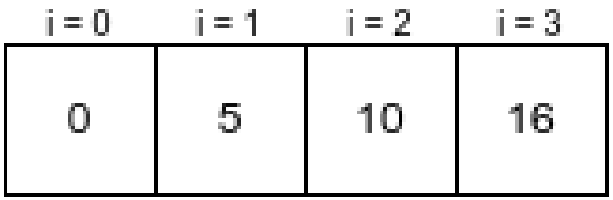


# Flight Controller: Domain decomposition




How many grid cells are required for a process?

- Rank $0 \rightarrow 5 - 0 = 5$ Starts 0
- Rank $1 \rightarrow 10 - 5 = 5$ Starts 5
- Rank $2 \rightarrow 16 - 10 = 6$ Starts 10



# Flight Controller: Reading planes with MPI

What is expected in the function `read_planes_mpi`?

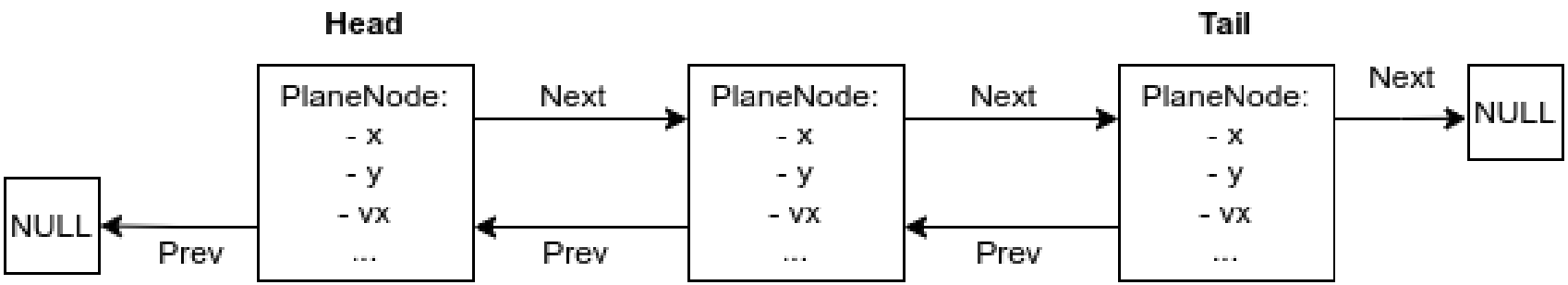
1. Review and understand the sequential reading.
2. Use the sequential version as a base code.
3. Distribute the grid cells among the processors using the variable: `tile_displacements`
4. For each rank, only the planes that are in its domain should be inserted to `PlaneList`.

# Flight Controller: Reminder double-linked list

Planes are stored by every process in a **double-linked list**.
Although, its canonical functions are implemented, such as:

- `insert_plane`
- `remove_plane`
- `seek_plane`

It's key to remember how to work with the list:



# Flight Controller: Communication Send/Recv

What is expected in the function `communicate_planes_send`?

1. Count how many planes I have to send, and to who? HINT: `get_rank_from_index`
2. Communicate with the group what I need to send and what I have to receive.
3. Send planes that are out of my domain and remove them from my current list. HINT: allocate buffers for an async send.
4. Receive planes from the rest of the group and add them to my list.

# Flight Controller: Async Send buffers

In a Non-blocking MPI Send, the application sending buffers should remain untouched until the communication finishes... Then, we have to allocate memory for every rank we are sending planes to: