Exercises of Parallelism - MPI

For all exercises consider that

* All useful imports (numpy, mpy4py, …) are already done
* rank and size have already been initialized to the rank and number of processes
* np.arange(a,b, dtype=’i’) produces an array of int elements from a included to b excluded
* np.zeros(nb, dtype = ‘f’) produces an array with nb float elements initialized at 0
* np.array([...]) converts the list into an array with the ‘correct’ type. np.array([1,2,3]) produces an array of int.

# Exercise 1 - Introduction

What is printed by the following code with 3 processes? Make a schema of the memory of each 3 processes (including A, rank, result, size).

A = np.zeros(2, dtype='f')

if rank == 0:

A[1] = 42

comm.**Bcast**(A, root=0)

**print** ("A on ", rank, "=", A)

result = np.empty(6, dtype = 'f')

comm.**Allgather**(A, result)

**print** ("AllGather", rank, "=", result)

A = A+1 # for numpy array, addition is element-wise

comm.**Gather**(A, result, root=2)

**print** ("Gather", rank, "=", result )

# Exercise 2 - Introduction

What is printed by the following code with 3 processes? Make a schema of the memory of the 3 processes: A, local\_A, result, …

if rank == 0:

A = np.arange(0, 6, dtype='f')

else:

A = None

local\_A = np.zeros(2, dtype = 'f')

comm.**Scatter**(A, local\_A, root=0)

**print** ("local\_A on ", rank, "=", local\_A)

local\_A = local\_A \* 3 # numpy : the multiplication is element-wise

result = np.zeros(2, dtype='f')

comm.**Reduce**(local\_A, result, op=MPI.SUM, root=1)

**print** ('Reduced on 1 : rank =', rank, 'result = ', result)

comm.**Reduce**(local\_A, result, op=MPI.MAX, root=0)

**print** ('Reduced on 0 : rank =', rank, 'result = ', result)

# Exercise 3

We have an image (array of int of values between 0 and 255) in a variable named pixels of length len(pixels). This image is only in the process with the rank==0. First assume the number of pixels can be divided by the number of processes. We want to count the number of pixels with a value strictly over 100.

1. What are the two collectives needed to distribute the pixels and obtain the final result **[collective choice]**
2. Write the pixel distribution completely using only collectives (i.e. with allocation of memory using np.empty when needed) **[memory management]**
3. Write the collective used to obtain the final result completely **[memory management]**

# Exercise 4

We suppose that the process of rank==0 has two int vectors **X** and **Y** of the same length **N**, of which it wants to make the scalar product **X.Y**. **X**, **Y** and **N** are known only by the process 0. N is divisible by the number of processes. We can modify X and Y if needed.

Reminder: [1,2,3] . [10, 20, 30] = 1\*10 + 2\*20 + 3\*30

We suppose that we have the basic function **scalarProduct**(X,Y) which returns the result of the scalar product of **X** by **Y**.

1. What are the collectives needed to distribute the values and obtain the final result **[collective choice]**
2. Write the first part of the code down to the use of the function **scalarProduct** completely (i.e. with allocation of memory) **[memory management]**
3. Finish the code so that the result is printed on rank==0. **[differentiated behavior]**
4. How would you change your code to remove the constraint of N divisible by the number of processes ? **[load balancing]**

# Exercise 5

We assume to have a vector **vec** of **N** real (i.e. float) numbers on P0, **size** processors, with N divisible by size.

1. Compute in parallel how many elements of the vector are greater than the mean. The calculation of the average must also be done in parallel. The final result is to be displayed by P0. **[complete example]**