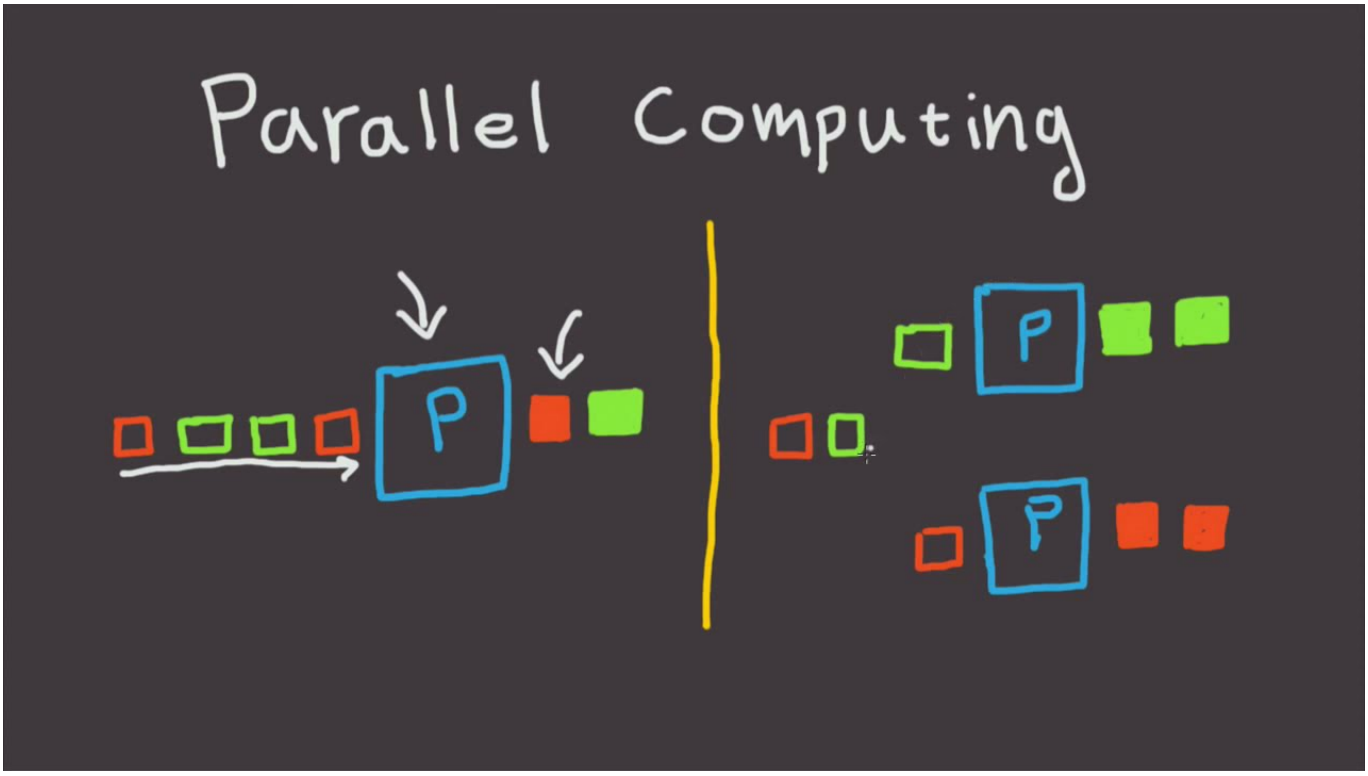


Tp1

- **Author: MITCHOZOUNOU Jean-CLaude**
  - **Institute: UM6P**

Assignment 1 - Point to Point communications

Exercise 1: Hello world

- **Let write an MPI program which prints the message "Hello World"**

```
from mpi4py import MPI
#Communicator , Rank and s i z e
COMM = MPI.COMM_WORLD
SIZE = COMM.Get_size()
RANK = COMM.Get_rank()
print("Hello World")
```

Output

```
(base) jc1er@jc1er-HP-EliteBook-Folio-1040-G3:~/Downloads$ cd JC_MPI
(MPI) jc1er@jc1er-HP-EliteBook-Folio-1040-G3:~/Downloads/JC_MPI$ mpirun -n 4 python Assignment_Exo1.py
Hello World
Hello World
Hello World
Hello World
(MPI) jc1er@jc1er-HP-EliteBook-Folio-1040-G3:~/Downloads/JC_MPI$
```

- **Let modify our program so that each process prints out both its rank and the total number of processes P that the code is running on.**

```python
from mpi4py import MPI

COMM = MPI.COMM_WORLD
SIZE = COMM.Get_size()
RANK = COMM.Get_rank()
print(f"Hello World {RANK} among {SIZE}")
```

Output

```
(MPI) jc1er@jc1er-HP-EliteBook-Folio-1040-G3:~/Downloads/JC_MPI$ mpirun -n 4 python Assignment_Exo1.py
Hello World 1 among 4
Hello World 2 among 4
Hello World 0 among 4
Hello World 3 among 4
(MPI) jc1er@jc1er-HP-EliteBook-Folio-1040-G3:~/Downloads/JC_MPI$
```

- **Let modify our program so that only a single controller process (e.g. rank 0) prints out a message**

```python
from mpi4py import MPI

COMM = MPI.COMM_WORLD
SIZE = COMM.Get_size()
RANK = COMM.Get_rank()
if RANK == 2: ## In our case we are choosen the
    print(f"Hello World {RANK} among {SIZE}")
```

Output

```
(MPI) jc1er@jc1er-HP-EliteBook-Folio-1040-G3:~/Downloads/JC_MPI$ mpirun -n 4 python Assignment_Exo1.py
Hello World 2 among 4
(MPI) jc1er@jc1er-HP-EliteBook-Folio-1040-G3:~/Downloads/JC_MPI$
```

Exercise 2 : Sharing Data

-

Let create a program that obtains an integer input from the terminal and distributes it to all the MPI processes. Each process must display its rank and the received value. Keep reading values until a negative integer is entered

```
from mpi4py import MPI

COMM = MPI.COMM_WORLD
SIZE = COMM.Get_size()
RANK = COMM.Get_rank()
Tag = 1


Tag = 1


while True:
    if RANK == 0:
            value = int(input("Please tape an integer "))
            if value < 0:
                for r in range(1,SIZE):
                    COMM.send(value, dest = r, tag = Tag)
                break
            print(f"Process {0} got {value}")
            for r in range(1,SIZE):
                COMM.send(value, dest = r, tag = Tag)
    else:
        data = COMM.recv(source = 0, tag = Tag )
        if data <0:
             break
        print(f"Process {RANK} got {data}")
```

Output

```
(MPI) jc1er@jc1er-HP-EliteBook-Folio-1040-G3:~/Downloads/JC_MPI$ mpirun -n 4 python Assignment_Exo2.py
Please tape an integer 6
Process 0 got 6
Please tape an integer 7
Process 0 got 7
Please tape an integer -3
Process 1 got 6
Process 1 got 7
Process 2 got 6
Process 2 got 7
Process 3 got 6
Process 3 got 7
(MPI) jc1er@jc1er-HP-EliteBook-Folio-1040-G3:~/Downloads/JC_MPI$
```

Exercise 3 : Sharing Data

- **Let mplement a ping-pong program using MPI. Process 0 sends an initial message (an integer) to process 1, which then increments the value and sends it back to process 0. This exchange happens for a fixed number of iterations**

```python
from mpi4py import MPI

COMM = MPI.COMM_WORLD
SIZE = COMM.Get_size()
RANK = COMM.Get_rank()
value = 0

N = 10 # Number of iterations

for i in range(N):
    if RANK == 0:
        COMM.send(value, dest=1)
        value = COMM.recv(source=1)
        print(f"Step {i}:I, process {RANK}, I received {value} from the
process 1.")
    print("\n")
    if RANK == 1:
        value = COMM.recv(source=0)
        print(f"Step {i}:I, process {RANK}, I received {value} from the
process 0.")
        value += 1
        COMM.send(value, dest=0)
```

Output

```
(MPI) jc1er@jc1er-HP-EliteBook-Folio-1040-G3:~/Downloads/JC_MPI$ mpirun -n 2 python Assignment_Exo3.py
Step 0:I, process 0, I received 1 from the process 1.
Step 1:I, process 0, I received 2 from the process 1.
Step 2:I, process 0, I received 3 from the process 1.
Step 3:I, process 0, I received 4 from the process 1.
Step 4:I, process 0, I received 5 from the process 1.
Step 5:I, process 0, I received 6 from the process 1.
Step 6:I, process 0, I received 7 from the process 1.
Step 7:I, process 0, I received 8 from the process 1.
Step 8:I, process 0, I received 9 from the process 1.
Step 9:I, process 0, I received 10 from the process 1.
Step 0:I, process 1, I received 0 from the process 0.
Step 1:I, process 1, I received 1 from the process 0.
Step 2:I, process 1, I received 2 from the process 0.
Step 3:I, process 1, I received 3 from the process 0.
Step 4:I, process 1, I received 4 from the process 0.
Step 5:I, process 1, I received 5 from the process 0.
Step 6:I, process 1, I received 6 from the process 0.
Step 7:I, process 1, I received 7 from the process 0.
Step 8:I, process 1, I received 8 from the process 0.
Step 9:I, process 1, I received 9 from the process 0.
(MPI) jc1er@jc1er-HP-EliteBook-Folio-1040-G3:~/Downloads/JC_MPI$
```

Exercise 4 : Sending in a ring (Broadcast by ring)

- **Let write a program that takes data from process zero and sends it to all of the other processes by sending it in a ring. That is, process i should receive the data add the rank of the process to it then send it to process i+1, until the last process is reached. Assume that the data consists of a single integer. Process zero reads the data from the user,print the process rank and the value received.**

```
from mpi4py import MPI

COMM = MPI.COMM_WORLD
SIZE = COMM.Get_size()
RANK = COMM.Get_rank()
#Tag = 1


if RANK == 0:
    data = int(input("Please tape an integer "))
    while data < 0:
        print("A value should be an integer")
        data = int(input("Please tape an integer "))
    print(f"I Process {0} got {data} from user JC")
    COMM.send(data, dest = RANK+1, tag = 0)

for i in range(1,SIZE-1):
     if RANK == i:
        data = COMM.recv(source = RANK-1, tag = 0)
        print(f"I Process {RANK} got {data} from Process {RANK-1}" )
        COMM.send(data+i, dest = i+1, tag = 0)

if RANK == SIZE-1:
    data = COMM.recv(source = RANK-1, tag = 0)
    print(f"I Process {RANK} got {data} from Process {RANK-1}")
```

Output

```
(MPI) jc1er@jc1er-HP-EliteBook-Folio-1040-G3:~/Downloads/JC_MPI$ mpirun -n 6 python Assignment_Exo4.py
Please tape an integer -4
A value should be an integer
Please tape an integer 20
I Process 0 got 20 from user JC
I Process 1 got 20 from Process 0
I Process 2 got 21 from Process 1
I Process 3 got 23 from Process 2
I Process 4 got 26 from Process 3
I Process 5 got 30 from Process 4
(MPI) jc1er@jc1er-HP-EliteBook-Folio-1040-G3:~/Downloads/JC_MPI$
```

Exercise 5 : Solving 1D advection equation

- **Let solve 1D advection equation using MPI**

```
import numpy as np
import matplotlib.pyplot as plt
from mpi4py import MPI
from time import process_time_ns
```

```
COMM = MPI.COMM_WORLD
nproc = COMM.Get_size()
RANK = COMM.Get_rank()

# Longueur du canal
L = 1000
# Temps Final
nt = 100
# Nombre de noeuds du maillage
nx = 200
# Le pas du maillage
dx = L/(nx-1)

# Vitesse du transport
c = 1
# CFL
CFL = 0.8
dt = CFL*dx/c


def f(x):
    if (300<=x<=400):
        return 10
    return 0

# Condition initiale
x = np.linspace(0,L,nx)
u0 = np.zeros(nx)
for i in range(nx):
    u0[i] = f(x[i])
# Tracé de la condition initiale
#plt.plot(x,u0,'-b')
#plt.grid()
#plt.show()




def solve_1d_linearconv(u, un, nt, nx, dt, dx, c):
    for n in range(nt):
        for i in range(nx): un[i] = u[i]
        for i in range(1, nx):
            u[i] = un[i] - c * dt / dx * (un[i] - un[i-1])
    return u

# Resolution parallèle

def solve_1d_linear_parallel(u, un, nt, nx, dt, dx, c):
    v = 0
    nloc = nx// nproc
    nlocP0 = nloc + (nx % nproc) # Au cas oú la division n'est pas exact ,
on étend le nombre
                # de noeuds du process 0 en y ajoutant le reste de la
```

```
division

    u1 = np.zeros(nlocP0+1)
    u2 = np.zeros(nloc+1)
    U = np.empty(0) # Pour collecter les sous sections de la solution sur


    #COMM.Barrier()

    # Initialisation des données sur chaque process


    if RANK == 0:
        for i in range(nlocP0):
            u1[i] = u[i]
        u1[nlocP0] = u1[nlocP0-1]
        COMM.send(u1[nlocP0], dest=RANK+1)
        #v = u1[0:nlocP0]


    for j in range(1,nproc-1):
        if RANK == j:
            u2[0] = COMM.recv(source = RANK-1)
            for i in range(nlocP0+(RANK-1)*nloc,nlocP0+RANK*nloc):
                u2[nlocP0+RANK*nloc-i] = u[i]
            u2[nloc] = u2[nloc-1]
            COMM.send(u2[nloc], dest=RANK+1)
            #v = u2[0:nloc]

    if RANK == nproc-1:
        u2[0] = COMM.recv(source = RANK-1)
        for i in range(nlocP0+(RANK-1)*nloc,nlocP0+RANK*nloc):
            u2[nlocP0+RANK*nloc-i] = u[i]
        #v = u2[0:nloc]



    # Calculs des valeurs u_n et échange des valeurs de bords

    for n in range(nt):
        if RANK == 0:
            for i in range(nlocP0): un[i] = u1[i]
            for i in range(1, nlocP0):
                u1[i] = un[i] - c * dt / dx * (un[i] - un[i-1])

            u1[nlocP0] = u1[nlocP0-1]
            COMM.send(u1[nlocP0], dest=RANK+1)
            v = u1[0:nlocP0]
            z = len(v)



        for j in range(1,nproc-1):
            if RANK == j:
```

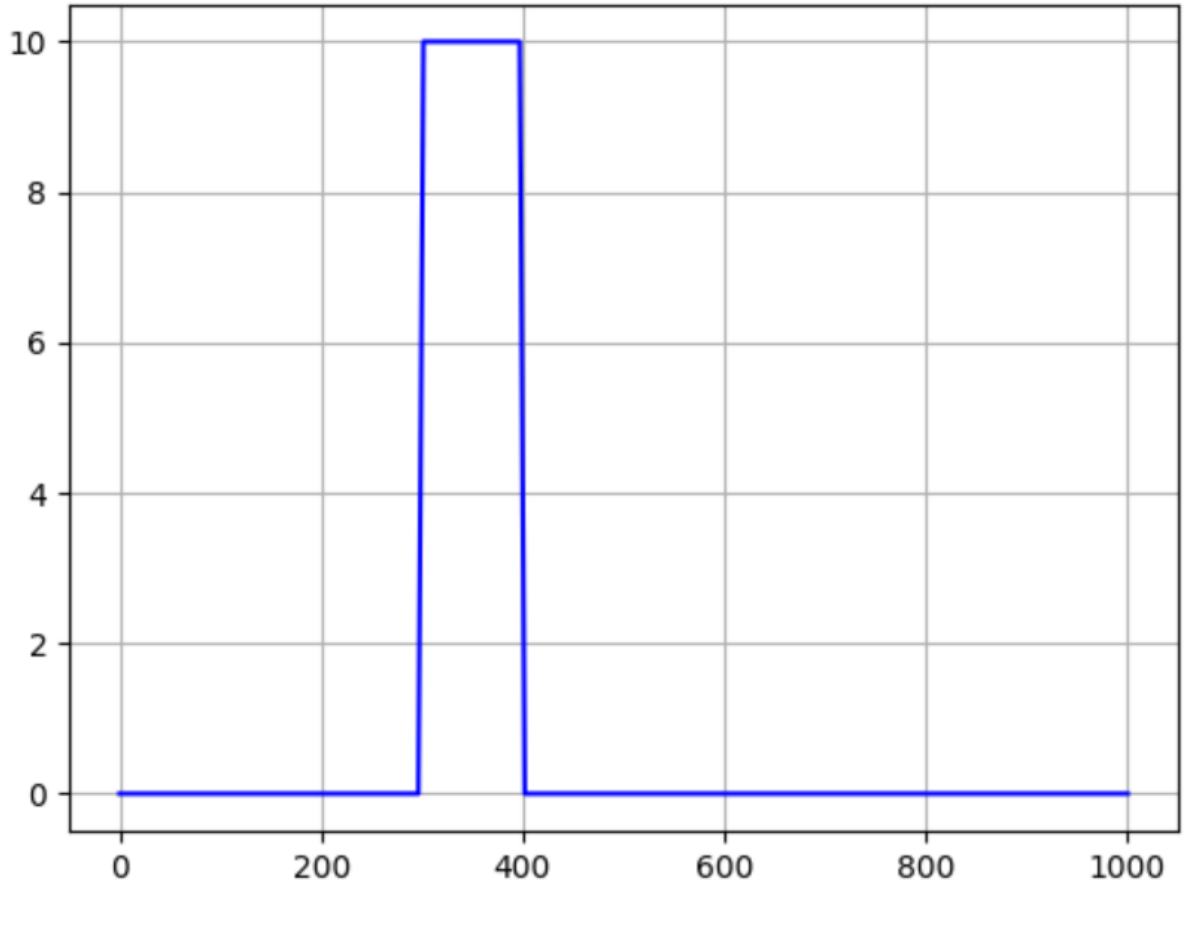
```
                for i in range(nloc): un[i] = u2[i]
                for i in range(1, nloc):
                    u2[i] = un[i] - c * dt / dx * (un[i] - un[i-1])
                u2[0] = COMM.recv(source = RANK-1)
                u2[nloc] = u2[nloc-1]
                COMM.send(u2[nloc], dest=RANK+1)
                v = u2[0:nloc]
                z = len(v)



        if RANK == nproc-1:
            for i in range(nloc): un[i] = u2[i]
            for i in range(1, nloc):
                u2[i] = un[i] - c * dt / dx * (un[i] - un[i-1])
            u2[0] = COMM.recv(source = RANK-1)
            v = u2[0:nloc]
            z = len(v)

        #COMM.Barrier()


# Racollement des vecteurs : On recolle tout via le process 0
    if RANK == 0:
        U = np.concatenate((U,v))
        for p in range(1,nproc):
            U = np.concatenate((U,COMM.recv(source = p, tag = 1)))
        #plt.plot(x,U)
        #plt.grid()
        #plt.show()
        #print("La solution s'écrit: U = ",U)

    for k in range(1,nproc):
        if RANK == k:
            COMM.send(v,dest = 0, tag = 1)

    return z #U




# Découpage des tâches et attribution aux processes.

if nproc ==1: # Le cas oú il y un seul processeur
    nloc = nx
    un = np.zeros(nx)
    u = solve_1d_linearconv(u0,un,nt,nx,dt,dx,c)
    plt.plot(x,u0,'-r')
    plt.grid()
    plt.show()

else: # Au moins deux processeurs
    un = np.zeros(nx)
    u = np.array(solve_1d_linear_parallel(u0,un,nt,nx,dt,dx,c))
```

```
print(f"Nombre de noeuds process{RANK} = {u}")
```

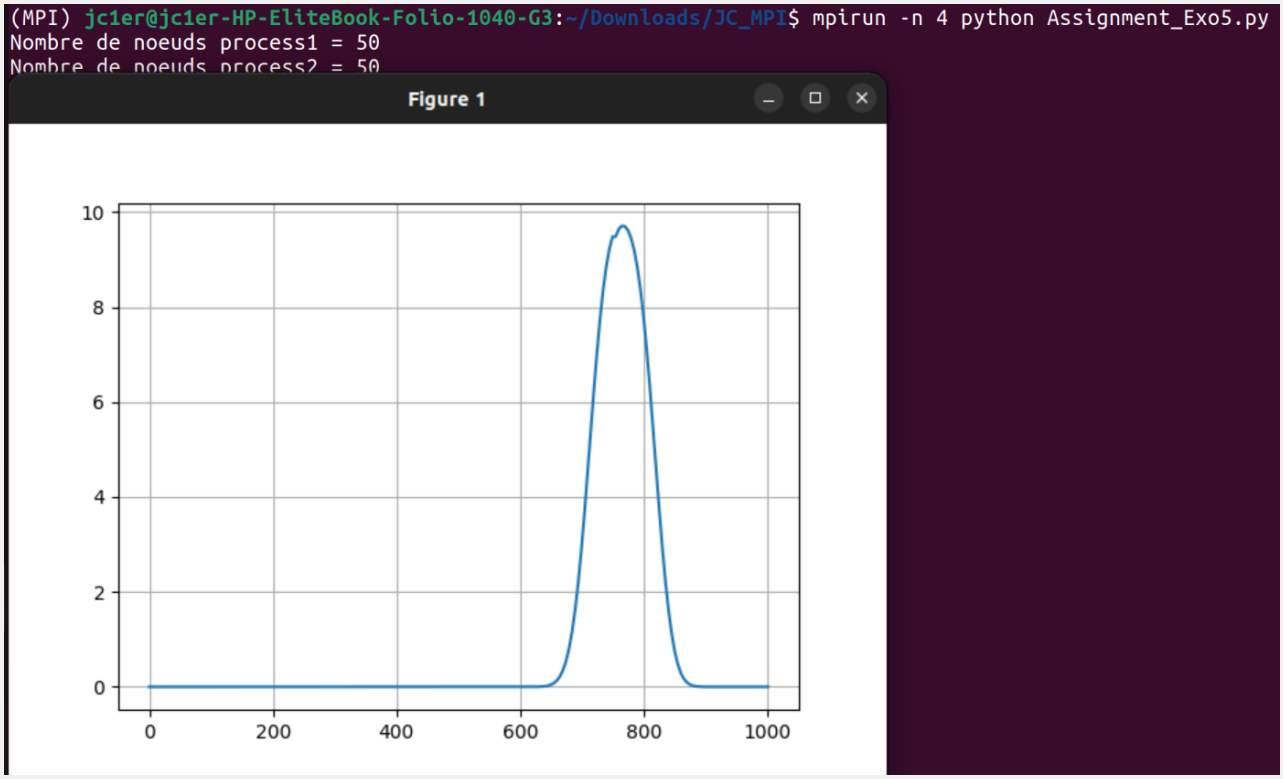
Gestion de la répartition du nombre de noeuds par processeur quelque soit le nombre process

```
(MPI) jc1er@jc1er-HP-EliteBook-Folio-1040-G3:~/Downloads/JC_MPI$ mpirun -n 4 python Assignment_Exo5.py
Nombre de noeuds process1 = 50
Nombre de noeuds process2 = 50
Nombre de noeuds process3 = 50
Nombre de noeuds process0 = 50
(MPI) jc1er@jc1er-HP-EliteBook-Folio-1040-G3:~/Downloads/JC_MPI$ mpirun -n 3 python Assignment_Exo5.py
Nombre de noeuds process1 = 66
Nombre de noeuds process2 = 66
Nombre de noeuds process0 = 68
(MPI) jc1er@jc1er-HP-EliteBook-Folio-1040-G3:~/Downloads/JC_MPI$ mpirun -n 6 python Assignment_Exo5.py
Nombre de noeuds process1 = 33
Nombre de noeuds process2 = 33
Nombre de noeuds process3 = 33
Nombre de noeuds process4 = 33
Nombre de noeuds process5 = 33
Nombre de noeuds process0 = 35
(MPI) jc1er@jc1er-HP-EliteBook-Folio-1040-G3:~/Downloads/JC_MPI$
```

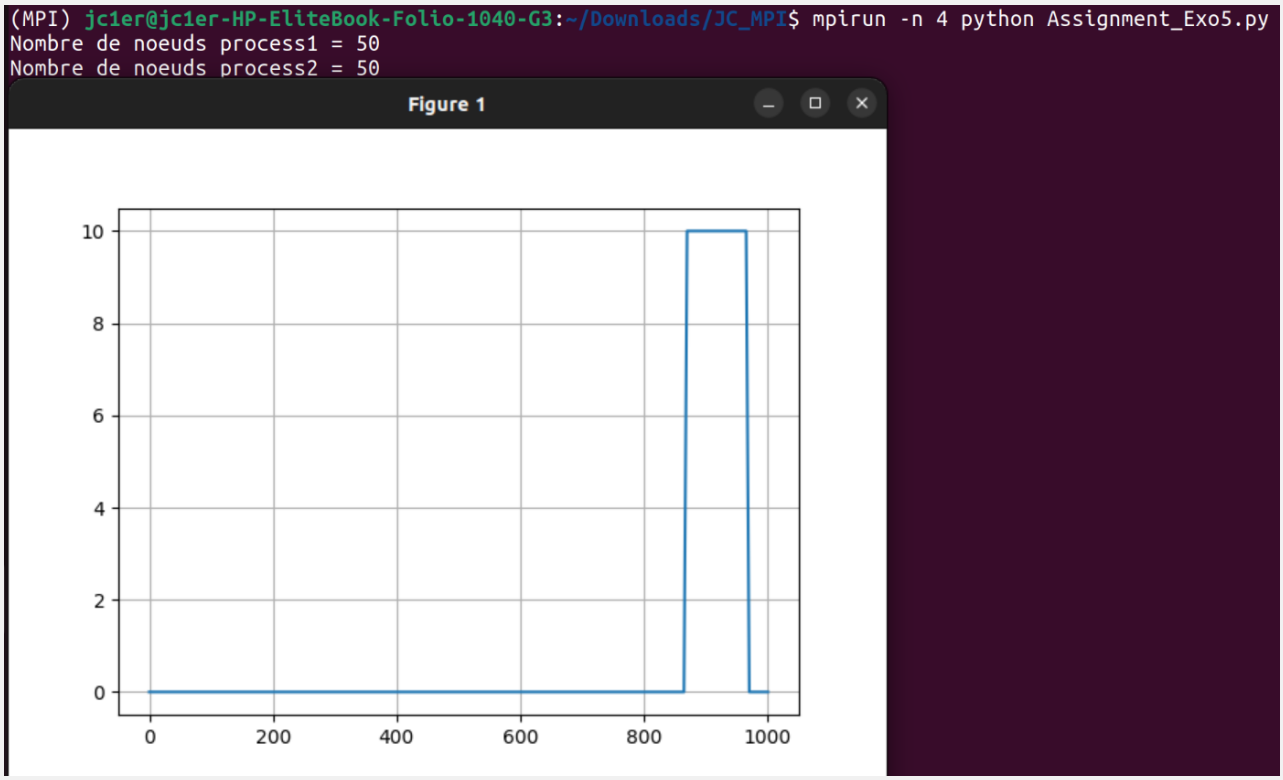
Solution initiale



Output : CFL = 0.7



Output : CFL = 1



Commentaire: On constate bien qu'il y une translation. Ce qui justifie bien la pertinence de la solution.

Lien vers les bouts de code