

Parallélisme

TD MPI

Elana Courtines
courtines.e@gmail.com
<https://github.com/irinacake>

Séance 1 - 03 octobre 2022

Ronan Guivarch - ronan.guivarch@irit.fr

Exercice 1 : Programme calculant le produit scalaire entre deux vecteurs

Note : il peut être intéressant de lister les étapes "à la main" avant de produire le code.

```
1 # python3 scalar_product.py 20
2 # mpirun -np 3 python3 scalar_product.py 13
3
4 import random
5 import sys
6 from mpi4py import MPI
7
8 def scalar_product(X,Y):
9     result = 0
10    for i in range(len(X)):
11        result+= X[i]*Y[i]
12    return result
13
14
15 # split a vector "x" in "size" part assuming that len(x)%size = 0
16 def split(x, size):
17     n = len(x) // size
18     return [x[n*i:n*(i+1)] for i in range(size)]
19
20 def split2(x, size):
21     n = len(x) // size
22     if len(x) % size != 0:
23         n+=1
24     return [x[n * i:n * (i + 1)] for i in range(size)]
25
26
27 if __name__ == '__main__':
28     # basic MPI variables
29     comm = MPI.COMM_WORLD
30     rank = comm.Get_rank()
31     size = comm.Get_size()
32
33     if rank == 0:
34         random.seed(0)          #"fixed" seed for random vectors
35         nb_elem = int(sys.argv[1]) # retrieve first argument as vector size
36         X = [random.random() for _ in range(nb_elem)]
37         Y = [random.random() for _ in range(nb_elem)]
38         Xsplit = split(X,size)
39         Ysplit = split(Y,size)
40     else:
41         Xsplit = None
42         Ysplit = None
43
44     local_X = comm.scatter(Xsplit, root=0)
45     local_Y = comm.scatter(Ysplit, root=0)
46
47     resultat = scalar_product(local_X, local_Y)
48
49     pscad = comm.reduce(resultat, op=MPI.SUM, root=0)
50
51     if rank == 0:
52         print("[",rank,"] Produit scalaire = ", pscad)
53         #version sequentielle :
54         print("[",rank,"] Produit scalaire sequentiel = ", scalar_product(X,Y))
```

scalar_product.py

Exercice 2 : Programme calculant la quantité d'éléments supérieur à la moyenne

```
# python3 scalar_product.py 20
2 # mpirun -np 3 python3 scalar_product.py 13
import random
4 import sys
from mpi4py import MPI

6
# split a vector "x" in "size" part assuming that len(x)%size == 0
8 def split(x, size):
    n = len(x) // size
    10 if len(x) % size != 0:
        n+=1
    12 return [x[n * i:n * (i + 1)] for i in range(size)]

14 if __name__ == '__main__':
    nb_elem = int(sys.argv[1]) # retrieve first argument as vector size
    16 # basic MPI variables
    comm = MPI.COMM_WORLD
    18 rank = comm.Get_rank()
    size = comm.Get_size()

    20
    if rank == 0:
        22 print("\nPartie 1 : calculer la moyenne\n")
        random.seed(0) # "fixed" seed for random vectors
        24 X = [random.random() for _ in range(nb_elem)]
        print("[", rank, "] X = ", X)
        26 Xsplit = split(X, size)
    else:
        28 Xsplit = None
        Ysplit = None

    30
    local_X = comm.scatter(Xsplit, root=0)
    32 print("[", rank, "] local_X = ", local_X)
    partial_mean = sum(local_X)/nb_elem
    34 global_mean = comm.allreduce(partial_mean, op=MPI.SUM)

    36
    if rank == 0:
        38 print("[", rank, "] Moyenne = ", global_mean)
        #version sequentielle :
        40 print("[", rank, "] Moyenne sequentiel = ", np.mean(X))
        print("\nPartie 2 : calculer la quantite d'element > moyenne\n")

    42
    partial_count = 0
    for i in range(len(local_X)):
        44 if local_X[i] > global_mean:
            partial_count+=1
    46 print("[", rank, "] partial_count = ", partial_count)

    48
    total_count = comm.reduce(partial_count, op=MPI.SUM, root=0)

    50
    if rank == 0:
        52 print("[", rank, "] Quantite = ", total_count)
        #version sequentielle :
        count = 0
        54 for i in range(len(X)):
            if X[i] > global_mean:
                count+=1
        56 print("[", rank, "] Moyenne sequentiel = ", count)
```

gt_mean.py