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.

```python
# python3 scalar_product.py 20
# mpirun -np 3 python3 scalar_product.py 13

import random
import sys
from mpi4py import MPI

def scalar_product(X,Y):
  result = 0
  for i in range(len(X)):
    result+= X[i]*Y[i]
  return result


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

def split2(x, size):
  n = len(x) // size
  if len(x) % size != 0:
    n+=1
  return [x[n * i:n * (i + 1)] for i in range(size)]


if __name__ == '__main__':
  # basic MPI variables
  comm = MPI.COMM_WORLD
  rank = comm.Get_rank()
  size = comm.Get_size()

  if rank == 0:
    random.seed(0)          #"fixed" seed for random vectors
    nb_elem = int(sys.argv[1])  # retrieve first argument as vector size
    X = [random.random() for _ in range(nb_elem)]
    Y = [random.random() for _ in range(nb_elem)]
    Xsplit = split(X, size)
    Ysplit = split(Y, size)
  else:
    Xsplit = None
    Ysplit = None

  local_X = comm.scatter(Xsplit, root=0)
  local_Y = comm.scatter(Ysplit, root=0)

  resultat = scalar_product(local_X,local_Y)

  pscad = comm.reduce(resultat, op=MPI.SUM, root=0)

  if rank == 0:
    print("[",rank,"] Produit scalaire = ", pscad)
    #version sequentielle :
    print("[",rank,"] Produit scalaire sequentiel = ",scalar_product(X,Y))
```

scalar_product.py

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

```python
# python3 gt_mean.py 20
# mpirun -np 3 python3 gt_mean.py 13
import random
import sys
from mpi4py import MPI

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

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

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

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

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

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

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

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

gt_mean.py