TP - MPI

Cet exercice consiste à implémenter une version parallèle des n-bodies, en MPI. Il reprend l'exercice 3 de la feuille de TD.

Le fichier de base s'appelle « *n-bodies-a-completer.py* »
Il se lance avec la commande :
 # mpirun -n 3 python3 n-bodies-a-completer.py 12 1000
où le premier paramètre est le nombre de corps (n-bodies), et le second sera le nombre d'iterations que nous souhaitons voir.

Il ne faut PAS toucher les instructions qui sont en début de fichier, tout le code que vous devez ajouter devra se trouver à la fin du fichier.
Le fichier fourni :

- fait les imports dont vous avez besoin
- définit un certain nombre de constantes et de fonctions utilitaires. Parmi celles-ci :
  - *init_world(n)* : cette fonction renvoie une liste contenant l'ensemble des n corps (n est passé en paramètre). Chaque corps possède plusieurs attributs : position, vitesse, poids. Cependant vous ne devriez pas avoir à les manipuler directement.
  - *update(d,f)* : cette fonction prend en paramètre un corps *d* et une force *f* et renvoie le corps *d* dont la position et la vitesse ont été modifiés par la force *f*
  - *interaction(corps1, corps2)* : cette fonction renvoie une liste à deux éléments représentant la force sur chacun des axes X et Y, résultante de l'interaction du corps1 sur le corps2
  - *display(m, l) :* cette fonction permet d'afficher les paramètres des corps contenus dans la liste *l* en préfixant cet affichage par le message *m*
  - *displayPlot(data) :* affiche les corps qui sont dans *data* sur la fenêtre graphique
  - *signature(world)* : calcul une valeur basée sur les caractéristiques des corps, qui représente la signature de ce monde. Utilisé après les itérations d'évolution, pour vérifier que l'évolution du monde calculée en séquentiel et en parallèle est la même.
  - *split(x, size)* : renvoie une liste qui est un découpage de la liste *x* en une liste de *size* listes. Potentiellement la dernière liste contient moins d'éléments que les autres, si ça ne se découpe pas juste.
  - *unsplit(x) :* crée une liste à partir d'une liste *x* de listes
  - les autres fonctions utilitaires et la classe n'ont pas vocation a être utilisées directement.

2.1 Commencer par écrire en python l'algorithme séquentiel, qui tournera donc seulement sur un processeur (lancement par *python3 n-bodies-a-completer.py 12 1000*). Noter la valeur de signature donnée pour le monde à la fin de la simulation.

2.2 En utilisant la bibliothèque MPI, proposez une version parallèle du code. Comparer la valeur de signature donnée pour le monde à la fin de la simulation avec la valeur séquentielle.

2.3 Ajouter dans le code de quoi mesurer le temps d'exécution (comm.Wtime()). Lancer l'exécution en utilisant 1, 2, 4, 8 instances, sur 1, 2, 4 et 8 processus (si possible sur votre machine) avec 1024 corps et 10 iterations. Tracer la courbe du speedup.