TD N°4 : **Multithreading avec Python**

Exercice 1

Soit le code suivant :

```
def square(x):
   return x*x

for i in [1, 2, 3, 4, 5, 6]:
 print(square(i))
```

1. Proposer une version de ce code permettant une exécution avec 6 threads, en utilisant le module _thread.
2. Proposer une version de ce code permettant une exécution avec 6 threads, en utilisant le module threading.
3. Proposer une version de ce code permettant une exécution avec 6 threads, en utilisant le module concurrent.futures.
4. Proposer une version de ce code permettant une exécution avec 3 threads, en utilisant le module concurrent.futures.

Exercice 2

Soit le code suivant qui permet d'avoir une approximation du nombre π, pour un nombre d'itérations égal à 10^6. A partir de ce code, il est demandé de proposer un autre qui permet d'avoir plusieurs approximations du nombre π pour différents nombres d'itérations. Ces approximations doivent se faire en parallèle avant d'être affichées. En effet, il est nécessaire de travailler avec 4 threads, tel que chaque thread calcule une approximation de π pour un nombre d'itérations différent et l'affiche.

```
import random

def sample(n):
    n_inside_circle = 0
    for i in range(n):
        x = random.random()
        y = random.random()
        if x**2 + y**2 < 1.0:
            n_inside_circle += 1
    return n_inside_circle

n_inside_circle = sample(10**6)

pi = 4.0 * (n_inside_circle /10**6)
pi
```

Exercice 3

Soit le code donné ci-après, donner sa version parallèle avec 6 threads. Il est exigé de soumettre du travail aux threads, en utilisant la méthode map().

```
import math

PREMS = [
112272535095293,
112582705942171,
112272535095293,
115280095190773,
115797848077099,
1099726899285419]

def est_prem(nb):
   if nb % 2 == 0: return False
   racine = int(math.floor(math.sqrt(nb)))
   for i in range(3, racine + 1, 2):
     if nb % i == 0: return False
   return True

def main():
  for i in PREMS:
      print('%d est premier? %s' %(i, est_prem (i)))

if __name__ == '__main__': main()
```

Exercice 4

Proposer un code permettant de calculer le produit scalaire de deux vecteurs *va* et *vb* de taille 100 000 en utilisant le parallélisme pour optimiser les performances. Pour cela, il est nécessaire de :

- Utiliser numpy pour effectuer le calcul du produit scalaire.
- Recourir à concurrent.futures et à ThreadPoolExecutor pour exécuter les tâches en parallèle.
- Créer deux threads pour le traitement.
- Soumettre les tâches aux threads à l'aide de la méthode submit().

Exercice 5

Soit le code suivant :

```
import numpy as np
from time import time

np.random.RandomState(100)
arr = np.random.randint(0, 10, size=[200000, 5])
data = arr.tolist()
data[:5]

def howmany_within_range(row, minimum, maximum):
    count = 0
    for n in row:
        if minimum <= n <= maximum:
            count = count + 1
    return count

results = []

for row in data:
    results.append(howmany_within_range(row, minimum=4, maximum=8))

print(results[:10])
```

1. Décrire ce que le code permet de réaliser.
2. Proposer une stratégie de parallélisation pour le code donné, en expliquant la répartition du travail entre les threads.

3. Présenter une version parallèle du code, utilisant 5 threads et intégrant la stratégie de parallélisation décrite, tout en faisant appel à la méthode submit() pour soumettre les tâches aux threads ?