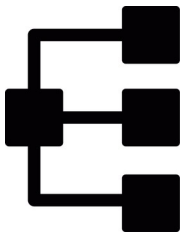
Parallélisation de tâches



Afin d'augmenter les performances des applications et de faire travailler un nombre maximum de cœurs d'une machine, il est possible de découper les traitements en une liste de sous-traitements (appelés tâches) et de les exécuter en parallèle plutôt qu'en séquentiel.

Une tâche est une procédure à exécuter qui peut attendre des paramètres en entrée et retourner un résultat. Les tâches parallèles sont entre autre utiles pour accélérer les temps de traitement de l'application. Plusieurs traitements sont alors exécutés en parallèle au lieu d'être exécutés séquentiellement : la vitesse de traitement est ainsi améliorée.

Objectif de la séance : reprendre l'algorithme du tri fusion et tirer partie de la méthode diviser pour régner pour effectuer ce travail avec des tâches parallèles.

1. Parallélisation des tâches de l'algorithme

Le programme incomplet de cet algorithme réparti sur plusieurs tâches parallèles, est fourni sur comparaison_tri_fusion.py

Q1. **Rappeler** le principe de l'algorithme de tri fusion ainsi que les deux fonctions `tri_fusion()` et `fusion()`

Q2. **Décrire** une méthode de parallélisation de cet algorithme.

On souhaite maintenant effectuer cette parallélisation en prenant comme nombre de processus deux fois le nombre de cœurs disponibles sur la plateforme, et où l'on partitionne les données à trier équitablement en paquets.

La bibliothèque `Multiprocessing` propose un ensemble de fonctions utiles pour la réserver et gérer des tâches parallèles dont `cpu_count()` qui renvoie le nombre de cœurs disponibles

Q3. **Déterminer** et **tester** une instruction qui renvoie le nombre de processus exécutables selon la machine utilisée. **Compléter** l'affectation de `nb_proc` avec cette expression (ligne 30).

Q4. On appelle `data` le tableau d'entiers à trier, **compléter** l'expression de la taille `np` des paquets qui seront triés par chaque processus.

`tdata` contient la liste des paquets à trier par les processus. La construction par compréhension du `tdata` défini ligne 32 est valable seulement si la longueur de la liste à trier est proportionnelle au nombre de processus.

Q5. **Modifier** la construction par compréhension du tableau `tdata` en considérant cette fois que la longueur de la liste à trier n'est pas forcément proportionnelle au nombre de processus.

La fonction fonction `tri_fusion_parallèle` trie chaque paquet avec un Pool du module `multiprocessing` qui permet de créer plusieurs tâches parallèles puis stocke les résultats dans une variable `pdata`.

Une liste ltest de 1000 données aléatoires désordonnées peut être obtenue par les commandes :

```
>>> import random
>>> ltest = [random.randint(0,10000) for _ in range(1000)]
```

Q6. **Exécuter** la fonction `tri_fusion_parallele` puis **décrire** et **justifier** le contenu de `pdata`.

Pour achever le tri, il faut fusionner les paquets de `pdata` 2 à 2 avec la fonction `fusion`. Cette fusion peut être effectuée par l'utilisation d'une file. Le principe consiste à copier `pdata` dans la file puis de `fusionner` les deux premiers éléments défilés dans une même liste et d'enfiler le résultat. Cette opération est répétée tant que la file contient plus d'un élément. La classe `file` du fichier `file.py` est disponible pour ce traitement.

Q7. **Compléter** la fonction `fusion_multiple()` à partir de la ligne 40, afin de fusionner avec la fonction `fusion()` les données contenues dans `pdata`. **Tester** la fonction `fusion_multiple()` puis **compléter** la fonction `tri_fusion_parallèle()` afin de fusionner les listes triées par chaque processus.

Q8. La fonction `tri_fusion_parallele()` doit maintenant être opérationnelle, **tester** la avec plusieurs jeux de données.

2.Etude des performances de la parallélisation des tâches

La fonction `performance()` permet de tester les performances de la parallélisation du tri de donnée.

Q9. **Etudier** la fonction `performance` et **décrire** le protocole de test de performance retenu.

Q10. **Exécuter** la fonction `performance` et **commenter** les performances respectives des différentes fonctions pour des jeux de données de taille allant de 2^8 à 2^{18}. **Discuter** de la performance de cette parallélisation et **identifier** son point faible.