RAPPORT MULTIPROCESSEUR

Lancement du fichier :

* Sous Windows : taper dans sa console >> set PYTHONPATH=.\jobs

>> python multi2.py .\jobs *n*

* Sous Linux : de la manière dans la console : >> export PYTHONPATH=’./jobs’

>> python multi2.py ‘./jobs’ *n*

On note *n* le nombre de processeurs que l’on récupère directement depuis la ligne de commande.

Fonctionnement du programme - Question 1 :

Tout d’abord on crée le log dans la fonction *run\_job()*, pour pouvoir y implémenter toutes les informations requises comme par exemple quel processeur a fait tourner quel fichier job. Ensuite on boucle le programme à l’aide de *multiprocessing.Pool* pour traiter tous les fichiers de type *job.py*. Une fois que le programme a fini de tourner, on a donc toutes les informations stockées dans le *app.log*, ainsi que les fichiers *job.result* dans le dossier ‘job’.

Le problème est que nous souhaitons pouvoir ajouter dans le log le temps total, ainsi que le nombre de processeur n qui a été imposé. C’est pour cela que nous déplaçons le fichier *app.log* dans le dossier supérieur (ici ‘MULTTT’), afin d’y ajouter les dernières informations nécessaires.

Enfin, une fois le premier programme *multi2.py* terminé, on pourra lancer le programme *annexe.py*, qui nous indiquera dans un nouveau document *axe.txt* dans quel ordre et combien de fois chaque processeur a été utilisé.

On analyse alors les résultats obtenus qui sont les suivants :

On constate tout d’abord que le nombre de processeurs de mon ordinateur est 6, et donc que … est 12. On remarque aussi assez facilement que plus il y a de processeurs et moins le temps est long, mais qu’il y a un grand gap à partir de 6 processeurs (en effet on passe de 57sec à 37sec).

En regardant les occurrences dans le cas n=9 présenté ci-dessus, on constate aussi que , . Tout cela nous amène à la conclusion suivante : .