SYSTÈMES AVANCÉS

Matthieu DEZERCES
Abdoul Basti MUKAILA ALH SHITTU

Université Paris Cité

2024

SOMMAIRE

PROGRAMMES

BENCHMARK

PROGRAMMES

Les 2 ordonnanceurs ont été développé. L'ordonnanceur par *work stealing* utilise une *Deques* en tant que liste doublement chaînée.

Le Make file permet plusieurs commande :

- **make sharing** : compile l'ordonnanceur par *work sharing* et créer le programme dans /bin
- **make stealing**: compile l'ordonnanceur par *work stealing*et créer le programme dans /bin
- **make all**: compile les 2 ordonnanceurs et créer les programmes dans /bin
- **make clean_obj**: clean le dossier /obj
- **make clean_bin**: clean le dossier /bin
- **make clean**: clean les dossiers /obj et /bin

Le programme benchmark.py permet d'exécuter les 2 ordonnanceurs et de créer un graphe selon leurs temps d'exécution.
Il se lance à l'aide de python et il faut avoir installé *matplotlib*.

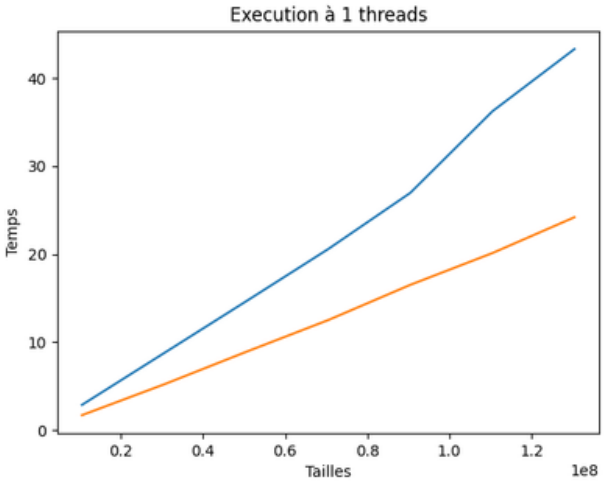
python3 benchmark.py [-t threads] [-n min_size] [-N max_size] [-e step]

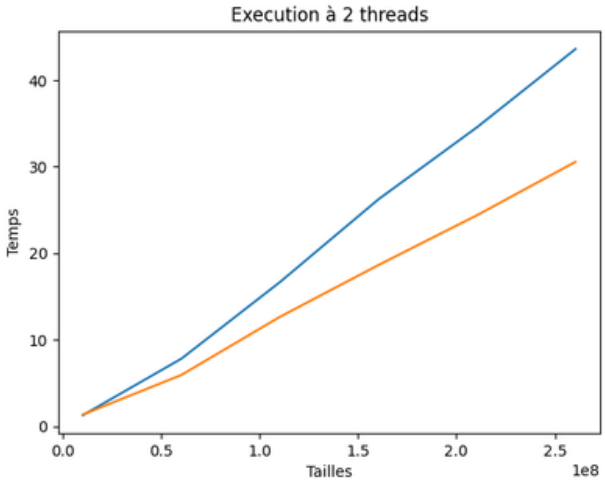
Avec:

- **threads**: le nombre de threads à créer.
- **min_size**: la taille du *quicksort* lors de la première exécution 10*1024*1024 par defaut.
- **max_size**: la taille maximal du *quicksort* lors de la dernière exécution 100*1024*1024 par defaut.
- **step**: la valeur d'augmentation de la taille du *quicksort* 10*1024*1024 par defaut.

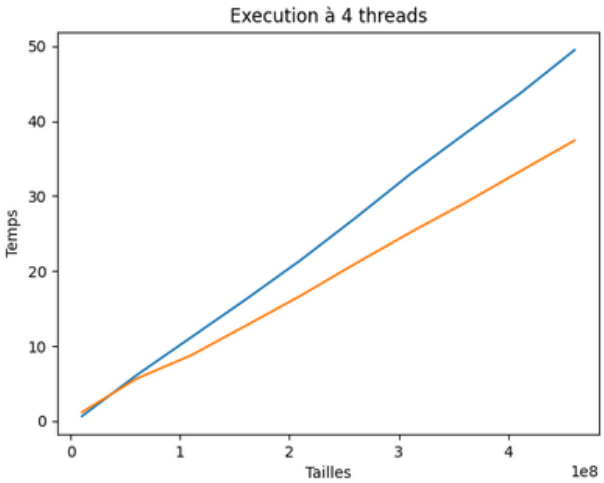
BENCHMARCK

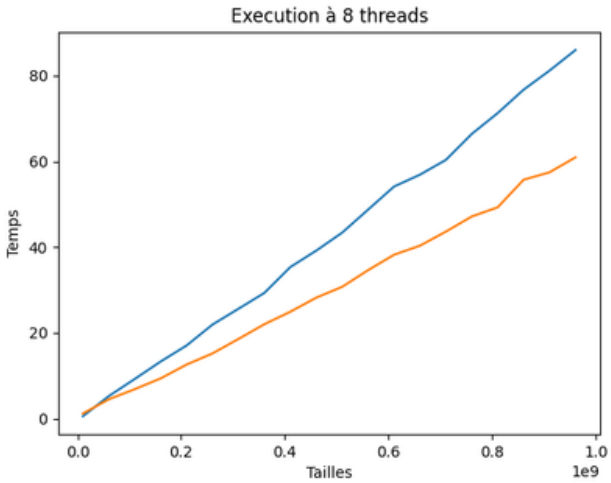
Les testes ont été effectué sur un sous système linux d'un windows possédant un processeur 11th Gen Intel(R) Core(TM) i5-1135G7 qui a4 cœurs soit 8 threads.





- *work sharing*
- *work stealing*





Les testes, même si insuffisant, nous montrent de meilleur performance du *work stealing* lors d'une augmentation du nombre d'exécution à faire.
Le *work sharing* est quand à lui plus efficace sur des petites données et s'améliore plus en fonction du nombre de threads.