Parallélisme intra-processeur - OpenMP

Exercice 2 - Premier test

Écrire un programme C qui se divise en 4 tâches openMP, chacune affichera son numéro de rang

```
valentin@MSI:~/m2/architecture_parallele/prog_parallele$ ./exo_2_1.out
threadNum = 2
threadNum = 1
threadNum = 0
threadNum = 3
```

Exercice 3 - Variable privé

Modifiez private par firstprivate et observer le résultat – déduction ?

private ne donne pas la garantie que la valeur est bien initialisé quand les Thread démarre tandis que la directive *firstprivate* nous assure que la valeur est initialisé

private clause

```
valentin@MSI:~/m2/architecture_parallele/prog_parallele$ ./exo_2_2.out
threadNum = 0 VALEUR_1 = 1000 VALEUR_2 = 32566
threadNum = 1 VALEUR_1 = 1000 VALEUR_2 = 32566
threadNum = 2 VALEUR_1 = 1000 VALEUR_2 = 32566
threadNum = 3 VALEUR_1 = 1000 VALEUR_2 = 32566
```

firstprivate clause

```
valentin@MSI:~/m2/architecture_parallele/prog_parallele$ ./exo_2_2.out
threadNum = 0 VALEUR_1 = 1000 VALEUR_2 = 2001
threadNum = 1 VALEUR_1 = 1000 VALEUR_2 = 2001
threadNum = 2 VALEUR_1 = 1000 VALEUR_2 = 2001
threadNum = 3 VALEUR_1 = 1000 VALEUR_2 = 2001
```

Exercice 4 - Boucles parallèles

Écrire un programme non parallèle qui compte de 0 à 50 en utilisant une simple boucle, et fait un printf du compteur à chaque itération.

Un thread

```
valentin@MSI:~/m2/architecture_parallele/prog_parallele$ ./exo_3.out
 i = 0 i = 1 i = 2 i = 3 i = 4 i = 5 i = 6 i = 7 i = 8 i = 9 i = 10 i = 11 i = 12 i = 13 i = 14 i = 15 i = 16 i = 17 i = 18 i
 i = 33 i = 34 i = 35 i = 36 i = 37 i = 38 i = 39 i = 40 i = 41 i = 42 i = 43 i = 44 i = 45 i = 46 i = 47 i = 48 i = 49 valen
```

Le temps moyen affiché est 0 car 50 est trop petit pour véritablement voir une différence entre threadé et non threadé

Modifier le code pour passer à 3 puis 4 tâches.

```
threadNum = 0 i = 12
threadNum = 2 i = 26
threadNum = 2 i = 27
threadNum = 2 i = 28
threadNum = 2 i = 29
threadNum = 2 i = 30
threadNum = 2 i = 31
threadNum = 2 i = 32
threadNum = 2 i = 33
threadNum = 2 i = 34
threadNum = 2 i = 35
threadNum = 2 i = 36
threadNum = 2 i = 37
threadNum = 3 i = 38
threadNum = 3 i = 39
threadNum = 3 i = 40
threadNum = 3 i = 41
threadNum = 3 i = 42
threadNum = 3 i = 43
threadNum = 3 i = 44
threadNum = 3 i = 45
threadNum = 3 i = 46
threadNum = 3 i = 47
threadNum = 3 i = 48
threadNum = 3 i = 49
threadNum = 1 i = 13
threadNum = 1 i = 14
```

4 threads

On peut voir que le comptage est bien répartis entre les 4 threads et mets en moyenne : *Average time 4 threads : 0.000664 seconds* (using WSL)

Exercice 5 - Calcul de PI

Note: j'ai réexécuté le calcul de PI en dynamique sur ma machine perso sous WSL et les valeurs obtenues sont incomparables avec les valeurs static sur les machines de l'ESIEE

1. Valeur de PI obtenue : *3.141592653589971*
2. Temps moyen calculé sur 10 itérations
    - PC ESIEE : 8 secondes
    - WSL

```
Using 1 thread PI=3.141592653589971 time = 2.378803
Using 1 thread PI=3.141592653589971 time = 2.386714
Using 1 thread PI=3.141592653589971 time = 2.371213
Using 1 thread PI=3.141592653589971 time = 2.376692
Using 1 thread PI=3.141592653589971 time = 2.395469
Using 1 thread PI=3.141592653589971 time = 2.391626
Using 1 thread PI=3.141592653589971 time = 2.411722
Using 1 thread PI=3.141592653589971 time = 2.381277
Using 1 thread PI=3.141592653589971 time = 2.373684
Using 1 thread PI=3.141592653589971 time = 2.414950
Average time using 1 threads: 2.388215 seconds
```

3. Temps moyen calculé sur 10 itérations sur 2 threads

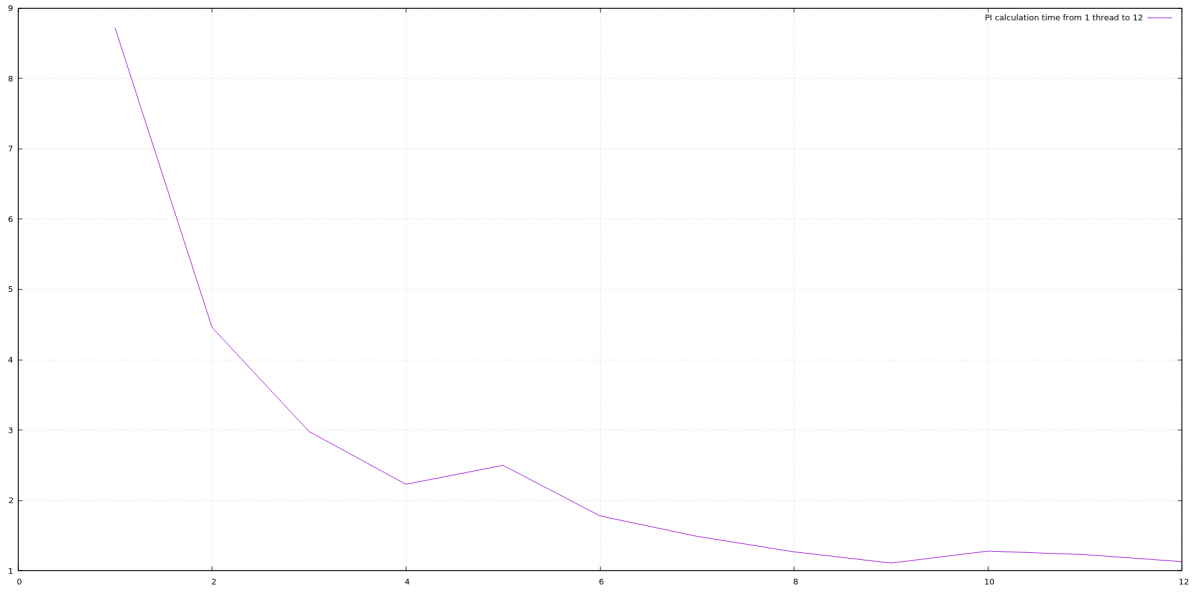
- PC ESIEE : `Temps d'exec moyen sur 10 essais avec 2 threads : 4.46 secondes`
- WSL :

```
Using 2 thread PI=3.141592653589901 time = 1.582937
Using 2 thread PI=3.141592653589901 time = 1.291830
Using 2 thread PI=3.141592653589901 time = 1.278759
Using 2 thread PI=3.141592653589901 time = 1.229700
Using 2 thread PI=3.141592653589901 time = 1.709470
Using 2 thread PI=3.141592653589901 time = 1.474050
Using 2 thread PI=3.141592653589901 time = 1.343033
Using 2 thread PI=3.141592653589901 time = 1.335888
Using 2 thread PI=3.141592653589901 time = 1.251783
Using 2 thread PI=3.141592653589901 time = 1.422735
Average time using 2 threads: 1.392018 seconds
```

4. On remarque que la durée d'exécution est divisé par 2 entre mono threadé et 2 threads

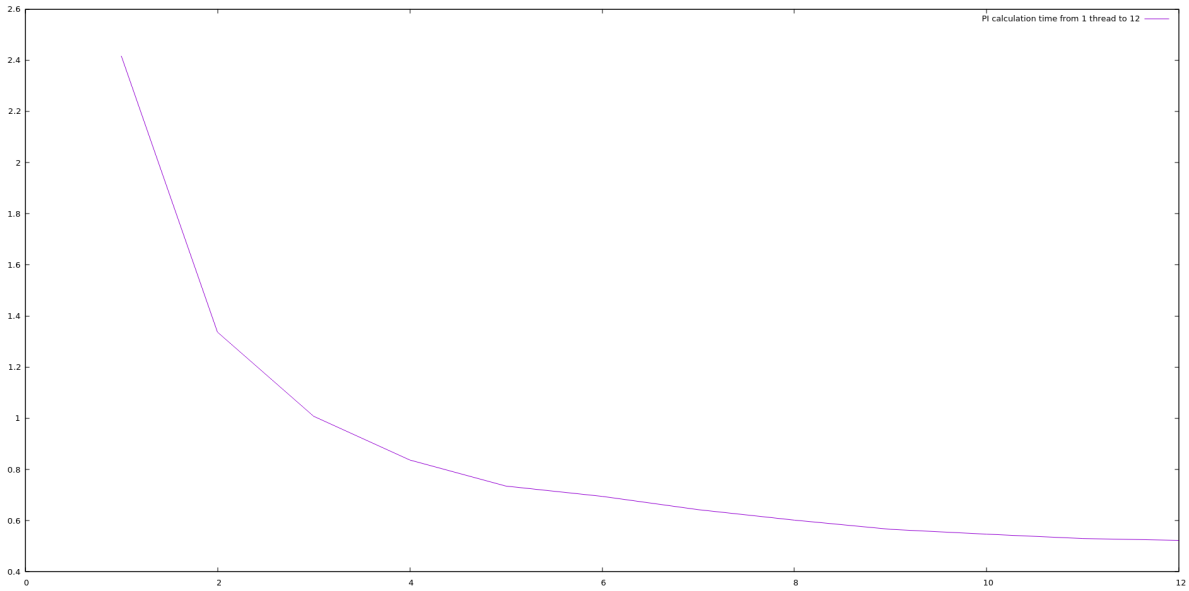
5. 12 threads

   - Machine ESIEE moyenne calculé sur 10 itérations :



     On peut voir que le temps moyen se divise par 2 jusqu'à 8 threads et se stabilise ensuite. Cela s'explique par le fait que 8 corresponds au nombre de coeur physique de la machine.

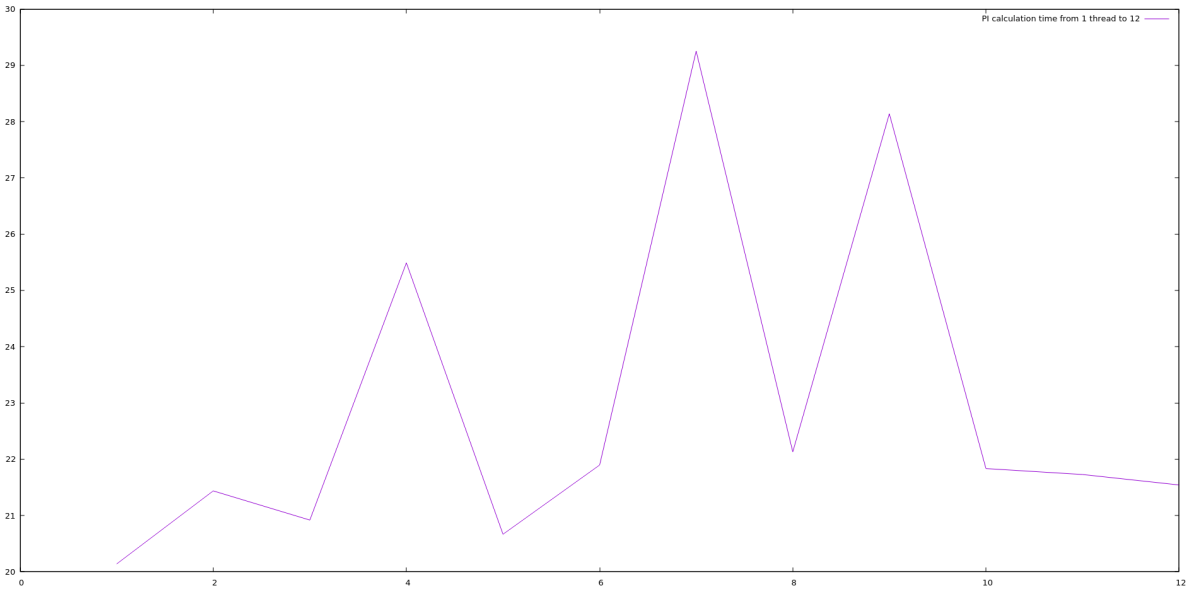
   - GNUplot WSL : moyenne calculé sur 10 itérations



La courbe présente la même allure mais l'intervalles de valeur est plus restreint que sur les machines de l'ESIEE. Cela peut s'expliquer par la différence de puissance de CPU.

6. Utilisation stratégie d'ordonnancement dynamic

   - GNUplot WSL : moyenne calculé sur 10 itérations

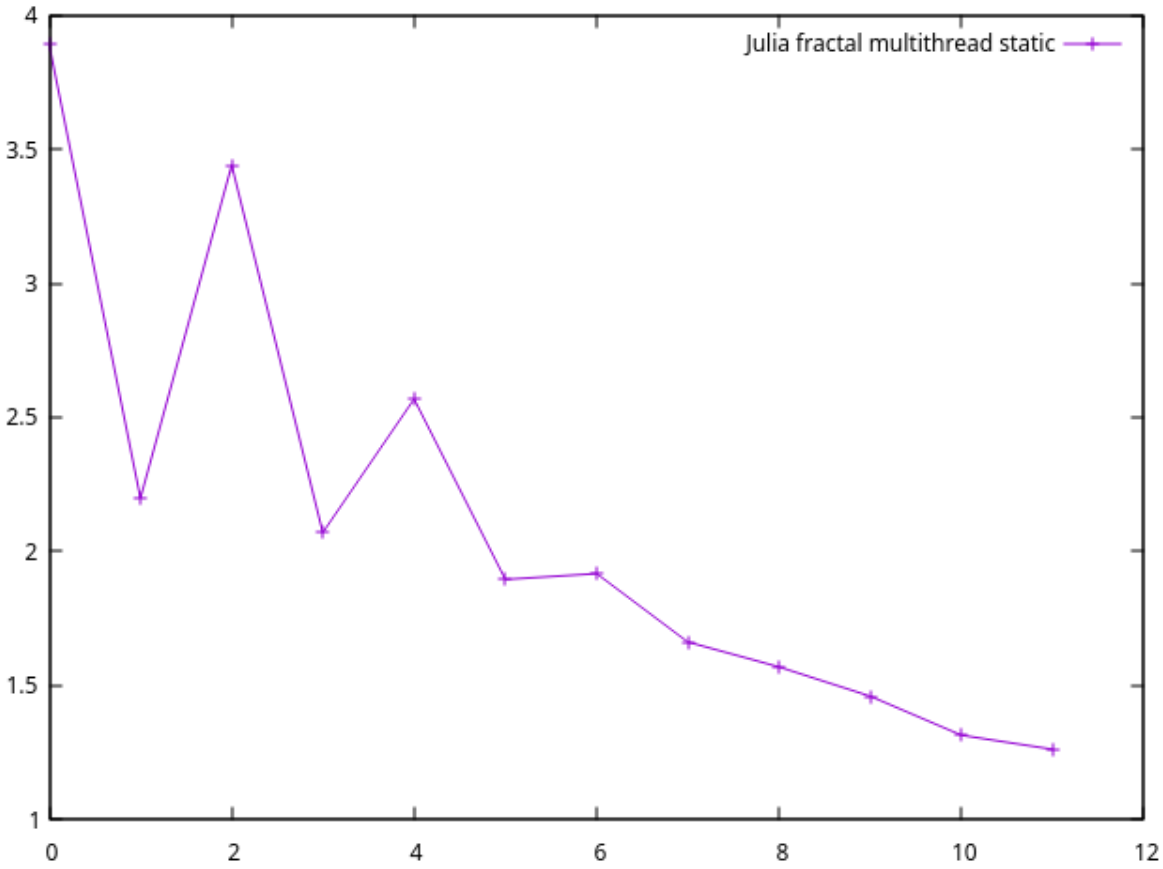


On peut constater que l'efficacité est bien moindre d'une part au vu des intervalles de valeurs. Aussi, l'allure de la courbe présentant des hautes amplitudes montrant l'inefficacité de la tâche en dynamic. Cela peut s'expliquer par WSL car c'est une VM et aussi à cause de l'homogénéité du calcul de PI. En effet, les threads auront une charge de travail similaire étant donné que c'est le même calcul.

Exercice 6 - Création de fractal, ensemble de Julia

4. Le temps d'exécution monothread, puis pour 2, 3, 4, 5, 6 etc threads en fonction du nombre de cœurs disponibles sur votre machine

- Static ESIEE



- Dynamic ESIEE