

# INF4033 Lab 6 :

## Pour aller plus loin avec les Pthreads

Alexandre BRIÈRE

## 1 Introduction

### 1.1 Objectifs

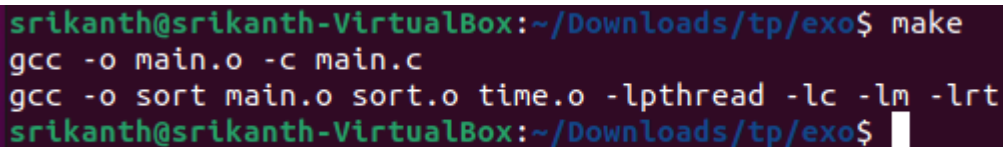
Mettre en pratique le paradigme "divide and conquer" grâce aux *Pthreads*.

Vous n'utiliserez pour ce TP que les fonctions présentes dans la bibliothèque *Pthread* (fork, sémaphore, etc. ne sont pas autorisés).

### 1.2 Pour commencer

Les codes sources sont fournis avec un makefile, tapez la commande suivante pour compiler :

```
$ make
```



```
srikanth@srikanth-VirtualBox:~/Downloads/tp/exo$ make
gcc -o main.o -c main.c
gcc -o sort main.o sort.o time.o -lpthread -lc -lm -lrt
srikanth@srikanth-VirtualBox:~/Downloads/tp/exo$
```

### 1.3 Astuces

Pour connaître le prototype exact des fonctions de la bibliothèque *Pthread*, pensez à la commande `man`.

```
$ man pthread_create
```

Pensez bien à vérifier toutes les valeurs de retour de la bibliothèque *Pthread*.

## 2 Exercice

Le but de l'exercice est de réaliser un algorithme de tri parallèle. L'application de tri a de nombreuses ressemblances avec l'application recherche max. Chaque thread trie individuellement sa partie de tableau. La fonction `sort_partial` permet de faire ce travail. Une fois les sous-tableaux triés, les threads se regroupent deux à deux (*terminant* et *fusionnant*). Chaque thread *terminant* retourne son tableau trié au thread *fusionnant* associé. Chaque thread *fusionnant* attend la fin du thread *terminant* associé, puis commence à fusionner son tableau avec celui du thread attendu.

Pour la première itération, les threads *fusionnants* sont ceux dont l'index est multiple de deux, les autres threads étant *terminants*. Lors de la deuxième itération, les *fusionnants* sont multiples de quatre, les autres sont *terminants*. La troisième itération, les *fusionnants* sont multiples

de huit, etc.

## 2.1 Completez le code

En vous servant du code de recherche du max et en utilisant le code donné dans le dossier exo, faites en sorte que le programme trie le tableau avec plusieurs threads.

Pour compiler le code, utilisez la commande make. Le programme attend en paramètre le nombre de threads à lancer pour effectuer le tri.

## 2.2 Mesure de performance

Le programme écrit sur la sortie d'erreur le nombre de secondes passées à faire le tri. Tracez le temps d'exécution en fonction du nombre de threads.

Que constatez-vous ? Comment expliquez-vous cela ?

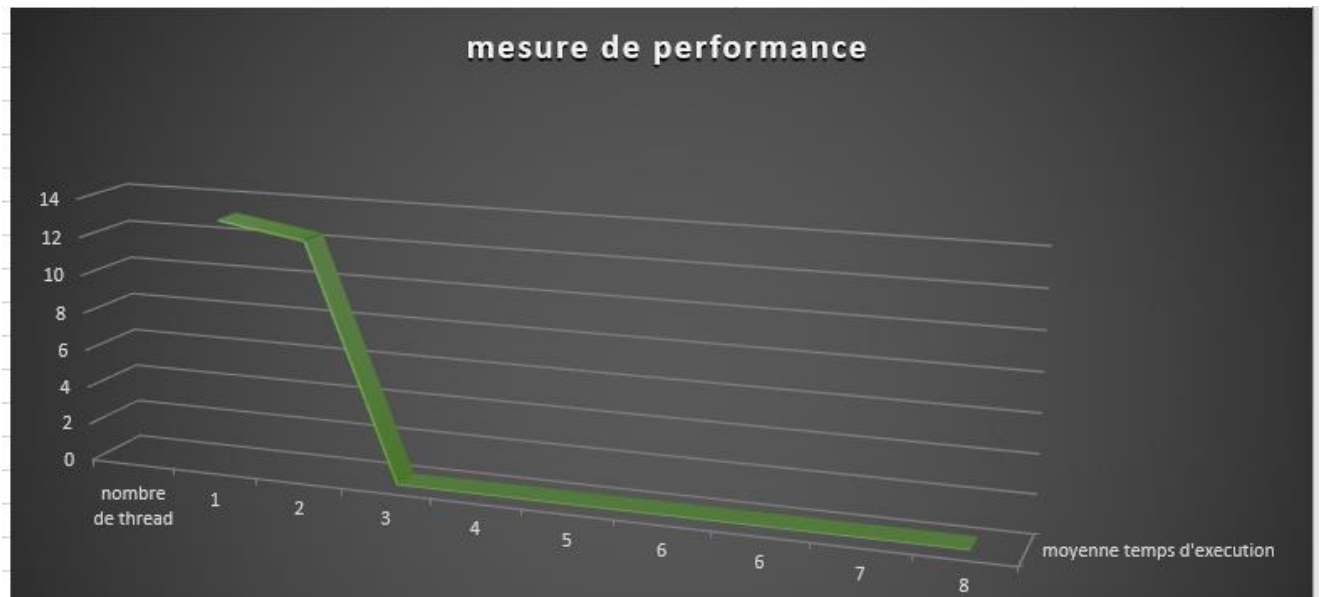
Dans un premier temps on a un tableau non trier puis un tableau trier (capture c-dessous)

```

srikanth@srikanth-VirtualBox:~/Downloads/tp/exo$ ./sort 1
tableau non trier
50 49 48 47 46 45 44 43 42 41 40 39 38 37 36 35 34 33 32 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5
4 3 2 1
tableau trier
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25

```

**Voici le graphique d'évolution en fonction du nombre de thread et du temps.**



**Explication et conclusion :** Avant de passer à 1000 threads, le processus prend moins de temps à s'exécuter avec plus de 1 threads qu'avec un seul thread, et le temps reste généralement constant par la suite. Mais le temps d'exécution devient plus considérable après 1000 car l'appareil n'a que 4 cœurs. Surtout on remarque que la courbe est non-linéaire.

Par exemple a 1000000 le temps devient beaucoup plus important, sauf dans mon cas l'ordinateur a du mal à afficher le temps d'exécution.

Cela s'explique par la faite que les tableaux sont divisés et repartie sur d'autre threads.

La performance de l'ordinateur joue également un rôle majeur sur le temps d'exécution.

Si on augmente le nombre de thread, on réduit le temps d'exécution car les thread se repartissent entre eux.

