CPOA : amphi 8 – Les Threads

Programmation concurrente

Avec les threads, il est possible d'exécuter plusieurs programme - ou partie de programme – en même temps en parallèle. Des interactions entre eux sont même possibles.

Il y a différentes applications de programmation concurrente à différentes échelles :

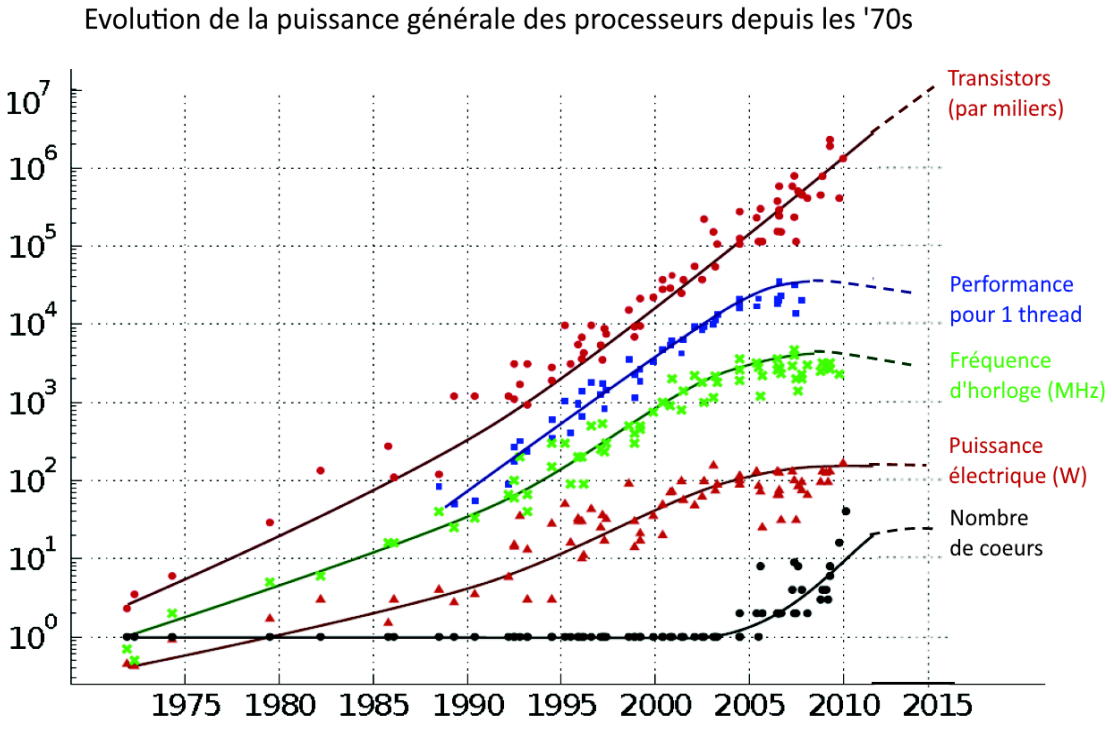
- Plusieurs ordinateurs dans un réseau (internet).
- Plusieurs applications sur un seul processeurs (sur un PC ; ils y a toujours plusieurs applications couvertes, à commencer par le GUI).
- Plusieurs processeurs dans un ordinateur (dans un jeu vidéo, le GPU s'occupe du calcul des éléments graphiques alors que le CPU s'occupe des autres calculs).
- Plusieurs cœurs dans un processeurs (ce qui permet de faire du calcul parallèle).

Exemple :

Pour un site web dit "synchrone", les appels potentiels à différents scripts vont se faire de façon procédurale, c'est-à-dire qu'ils vont s'exécuter les uns après les autres, en attendant à chaque fois la fin du script précédent pour lancer le suivant.

A l'inverse, sur un site web dit "asynchrone", les différents scripts vont pouvoir s'exécuter en parallèle, ce qui apporte un gain de temps, mais aussi des contraintes de programmation.

Processeurs



Original data collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond and C. Batten
Dotted line extrapolations by C. Moore

Depuis que les processeurs existent, leur puissance n'a cessé d'augmenter exponentiellement : on y met toujours plus de transistors, de cœurs, avec une fréquence toujours plus importante. Mais depuis une dizaine d'années, on note une continuité dans l'évolution de la fréquence des processeurs, on n'arrive plus à la faire augmenter. Le but de nos jours - pour continuer à faire évoluer la puissance générale des processeurs - est donc d'exploiter le parallélisme informatique avec plus de cœurs dans les CPU, mais c'est une tâche que s'avère difficile.

Processus vs Thread

Processus	Thread
Mémoires séparés entre différents processus, plus lourd à mettre en place.	Mémoire partagée entre différents threads, moins lourd à mettre en place.

Exemple de Threads

```
class Compte {
    private int solde = 0;
    public void deposer() {
        solde++;
    }
}
```

Méthode `deposer()` :

1. Lire solde
2. Calculer solde+1
3. 3. Ecrire solde ← solde+1

Exécution séquentielle :

- T1.1 : Lire solde = 0
- T1.2 : Calculer 0+1 = 1
- T1.3 : Ecrire solde = 1
- T2.1 : Lire solde = 1
- T2.2 : Calculer 1+1 = 2
- T2.3 : Ecrire solde = 2

Exécution parallèle :

- T1.1 : Lire solde = 0
- T1.2 : Calculer 0+1 = 1
- T2.1 : Lire solde = 0
- T2.2 : Calculer 0+1 = 1
- T2.3 : Ecrire solde = 1
- T1.3 : Ecrire solde = 1

Verrous

- En Java, tout objet peut être utilisé comme un verrou : `synchronized(obj) {…}`
- A chaque instant, au plus une section de code d'un même objet (verrou) peut être exécutée.
- S'il y a un autre thread qui possède le verrou, on doit attendre qu'il ne quitte la section `synchronized`.

Implémentation Java

En Java, il y a deux façons d'implémenter les threads pour sur classe : soit en faisant un `extends Thread`, soit en faisant un `implements Runnable`.