**Mail :** yassine.elkhalki@outlook.fr
**Nom :** ELKHALKI Yassine
**Groupe :** INF3-FA


IUT de Vélizy-Rambouillet
CAMPUS DE VÉLIZY-VILLACOUBLAY
CAMPUS DE RAMBOUILLET

**Dépôt Git :** https://github.com/YassouSensai/yassine_prog_avancee.git

---

**Ce rapport a été réalisé en se basant sur les différents CMs & TPs/TDs**

**Pour vérifier certaines informations, certaines requêtes ont été demandées à ChatGPT.**

# Sommaire

# Programmation Avancée

## Cours

### CM3

**Deux modèles de programmation parallèle :**

- Parallélisme de tâche : une tâche principale est décomposée en plusieurs sous tâches indépendantes (communication par message)
- Parallélisme de données : les données sont divisées en blocs qui seront traité par une tâche distincte

**Plusieurs paradigmes de programmation parallèle :**

Paradigme Master Worker :

- Master (Maître) : Coordonne le travail et distribue les tâches aux esclaves
- Worker (Esclave) : Exécute les tâches et renvoie les résultats au maître

Autres : Clients Serveur, itérations parallèles (queues en Java, pools en Python), …

**Architectures :**

- SISD (Single Instruction Multiple Data) : sur processeur monocoeur
- SIMD (Single Instruction Multiple Data) : GPU
- MIMD ( Multiple Instruction Multiple Data) : systèmes multi coeurs

- Mémoire partagée : Tous les processus accèdent à la même mémoire (multiprocesseur)
- Mémoire distribuée : Chaque processus a sa propre mémoire (cluster)
- Systèmes hétérogène : CPU +GPU

# CM4

**API concurrent :**

- Executor : Mécanisme qui permet de gérer les threads
- Future : Représente le résultat d'un calcul asynchrone (vérifie si la tâche est terminée et le récupère)

**Evaluation des performances :**

- Speedup (Sp)  (Accélération)  = Gain de vitesse obtenu en utilisant plusieurs processus par rapport à un seul. ***Sp = T1/Tp*** ou T1 est le temps d'exécution pour un seul processus et Tp pour plusieurs
- Scalabilité forte : on fixe la taille du problème tout en augmentant le nombre de processus. "Grossièrement", plus on a de processus et plus ça doit aller vite.
- Scalabilité faible : On augmente proportionnellement la taille du problème par rapport au nombre de processus.

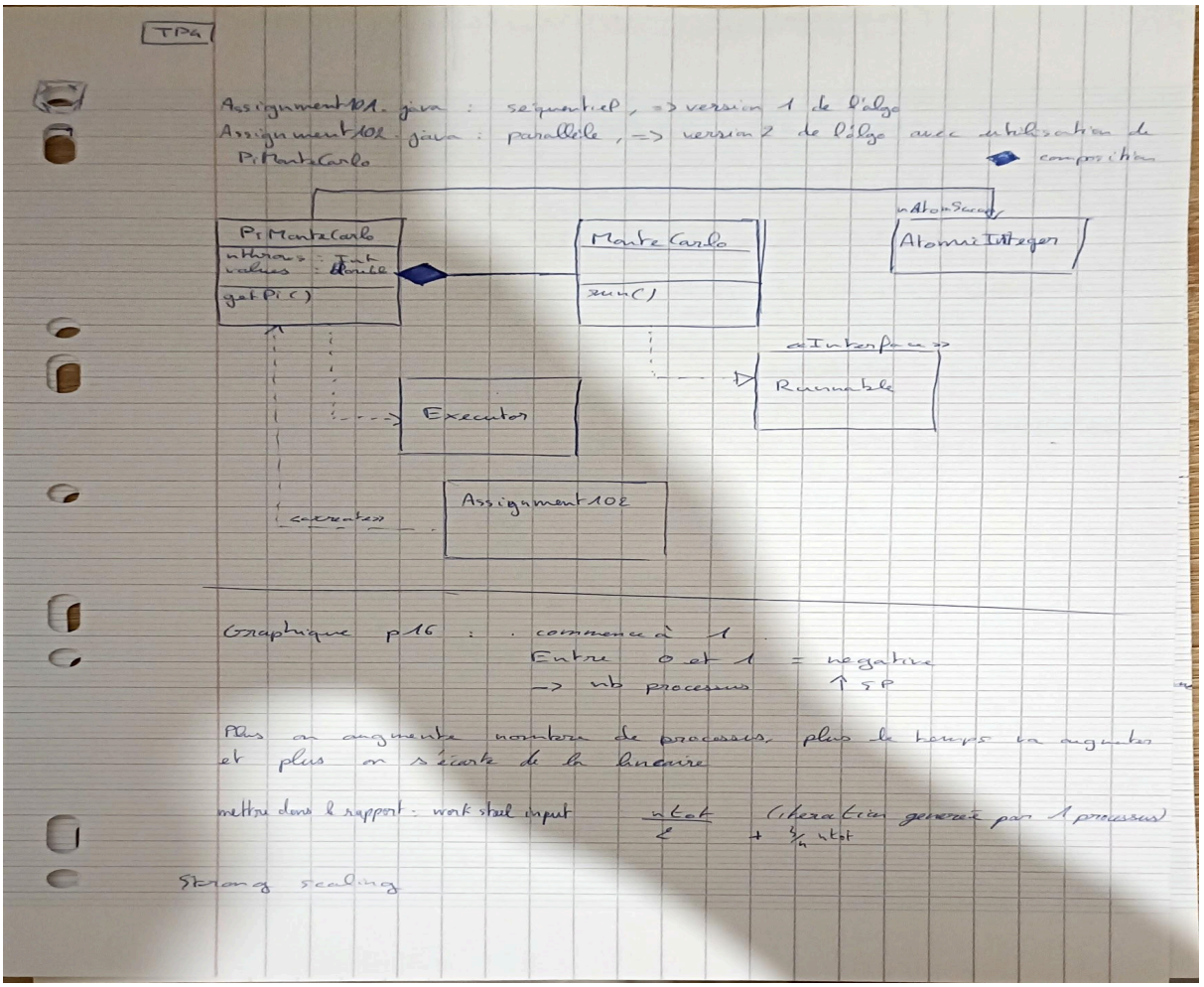
La scalabilité forte permet d'évaluer si l'on va plus vite avec plus de processus = Question de temps.

La scalabilité faible permet d'évaluer si l'on peut faire plus avec plus de processus = Question de "production" (Mon mot 😬)

# Evaluation du code source

## Pi et Assignement102

*A refaire au propre sur draw.io*




# Calcul de π

*Calcul de π par une méthode de Monte Carlo*

## Méthode de Monte Carlo

Les méthodes de Monte Carlo sont des techniques de simulation basées sur des tirages aléatoires pour résoudre des problèmes numériques, notamment en physique, finance et en optimisation.

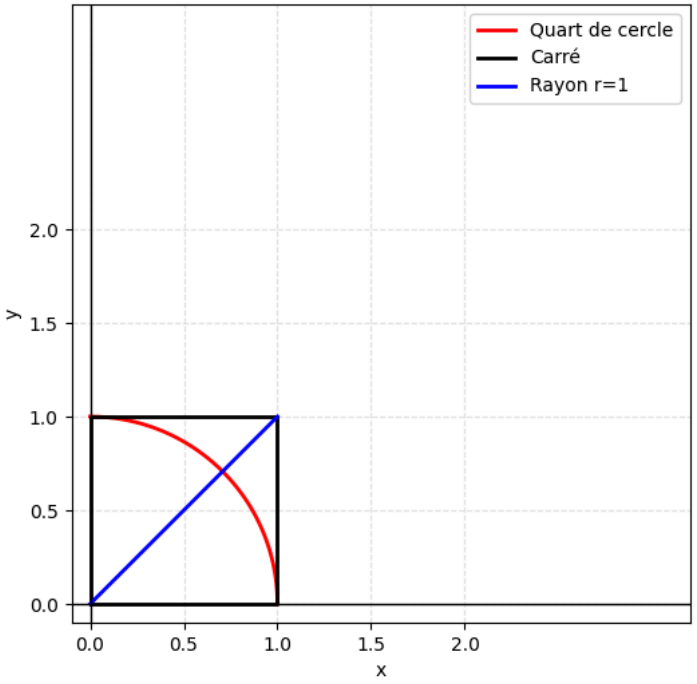
Exemples :

- Simulation de la propagation du feu (abordée en discussion en entreprise) (Physique)
- Estimation de la valeur future d'un actif (Finance)

- Utilisation d’algorithmes d’optimisation comme avec scipy en Python

## Détermination de l’algorithme de Monte Carlo pour π

*Partie réalisée en cours - Réécriture au propre - Voir Annexe*




- Soit un carré de côté 1
- Soit un quart de disque de rayon r = 1
- L’aire du carré s’écrit Ac = r^2 = 1*1 = 1
- L’aire du quart de disque s’écrit Ad4 = π*r^2 / 4 = π/4
- On effectue n_total tirages

En suite, on sait que la probabilité qu’un point Xi soit dans le quart de disque si sa valeur di est comprise dans le quart de disque :

$$P(Xi \mid di < 1) = Ad4 / Ac = (\pi/4) / 1 = \pi/4$$

*En sachant que chaque point Xi a deux coordonnées xi et yi et que sa valeur di est obtenue par* $di = \sqrt{( xi^2 + yi^2)}$

On sait également que plus n_total est grand, plus la valeur estimée de π sera proche de la valeur réelle de π.

On peut donc estimer la probabilité que Xi soit dans le quart de disque est :

$$P(Xi \mid di < 1) = n\_cibles / n\_total$$

*Avec n_cibles, le nombre de points qui se trouvent dans le quart de disque.*

Finalement, on peut déduire que la valeur approximative de π est :

$$\pi/4 = n\_cibles / n\_total$$
$$\Leftrightarrow \quad \pi = 4 * n\_cibles / n\_total$$

## L'algorithme en code

*Voir version manuscrite (prise de note)*

```
n_cible = 0;
for (p = 0; n_tot > 0; n_tot--) {
    x_p = rand();
    y_p = rand();
    if ((x_p * x_p + y_p * y_p) < 1) {
        n_cible++;
    }
}
pi = 4 * n_cible / n_tot;
```

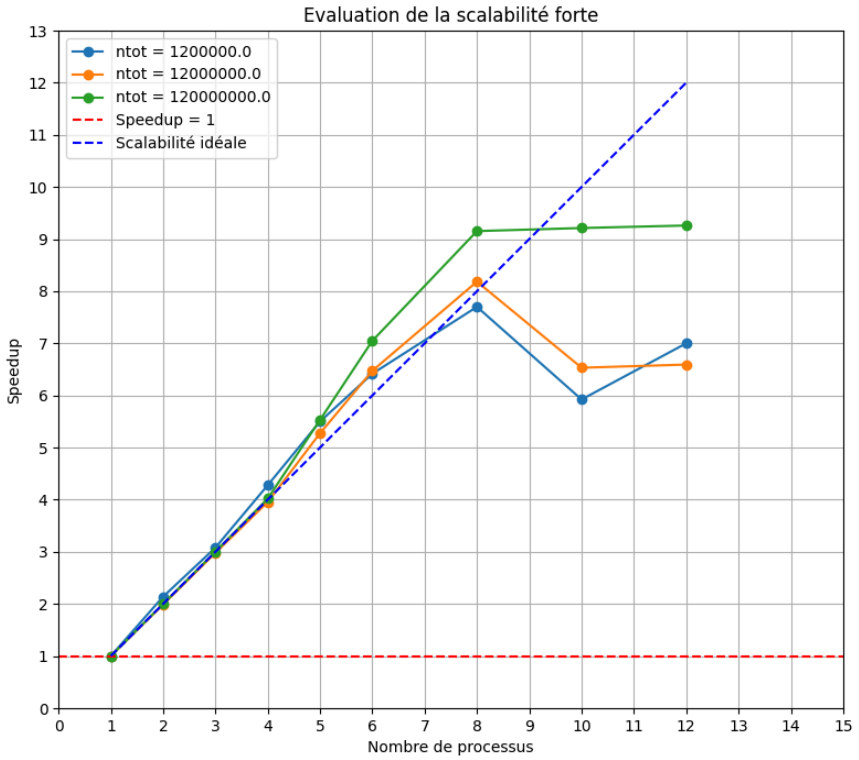
## Parallélisation - Paradigme Master Worker

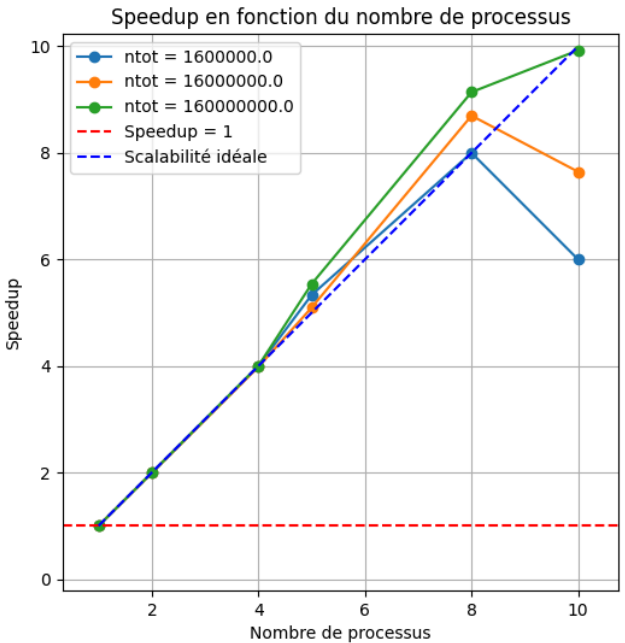
Ajout de UML plus explication des modifications

# Evaluation des performances (Mac M3)

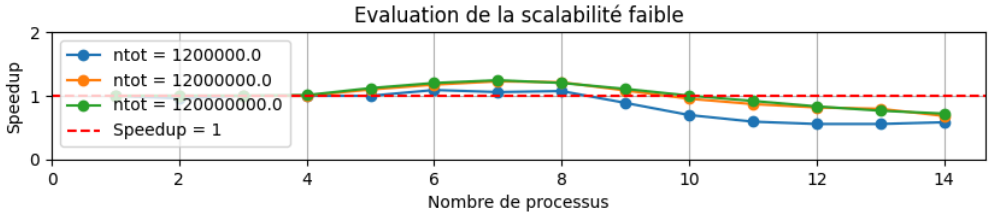
Sur puce M3 (8 coeurs CPU 10 coeurs GPU) (4 coeurs de performance, 4 coeurs d'efficacité)

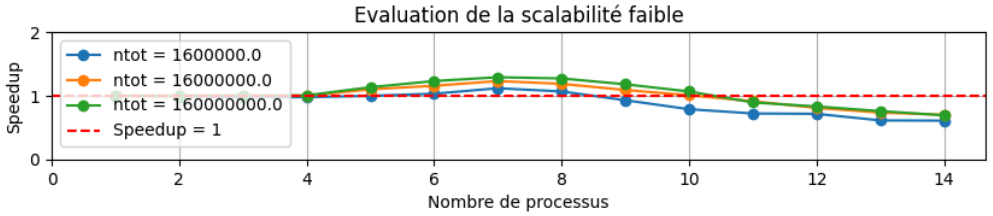
**Scalabilité forte :**



**Scalabilité faible :**




## Evaluation des performances (PC G26)

# Qualité de développement

## Notes pas à pas

SQuaRE : System and Software Quality Requirements and Evaluation
25010 : voir page 14 (system) pour les explications

Quality News : Experience Utilisateur
Quality Product : Technique

Pages 7 à-11 :

Functional suitability = A quel point je vérifie mes cas d'utilisation  (influence sur les utilisateurs principaux)

Performance efficiency = Performance du logiciel en fonction des ressources (matérielles : CPU/GPU, …) à disposition (influence sur les utilisateurs principaux + information de qualité qui concernent les différentes parties prenantes (clients ou autres) => gains de productivité)

Compatibility : Un composant peut utiliser/appeler un autre composant (important pour la maintenance et l'évolution)  => Conception architecturale 2eme année

Portability : Compatibilité d'un code d'une machine à une autre (influence sur la maintenance)

Maintainability : (justesse + efficacité)  Degré de modification (==Flexibilité) possibilité => N'influence pas l'expérience utilisateur car le rendu est sûrement le même. Pas les autres parties prenantes car ils ne sont pas intéressé par ça mais plutôt les résultats
(Ex : Il y a un bug => a quelle vitesses je peux trouver les bugs et les corrigés facilement)

Throughput rates : (débit) temps de communication d'un produit ou d'un système

Capacity : au maximum combien (ex : nombre de personnes maximum connectés sur un serveur)

[effectiveness : justesse
efficiency : efficacité]

Modularité : Degré avec lequel un système est composé de composants tel que si l'un est modifié, l'impact est moindre. => application modulaire.

Reusability : Réutilisation d'un composant par d'autres composants

Analysability : Couverture par des tests/prints

Modifiability : À quel point je peux modifier sans impacter les autres critères de qualité

## Important / À retenir