

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 et améliorer ce rapport, certaines requêtes ont été demandées à ChatGPT.

Sommaire

Programmation Avancée.....	2
Cours.....	2
CM3.....	2
CM4.....	3
Evaluation du code source.....	4
Pi et Assignement102.....	4
Calcul de π	4
Méthode de Monte Carlo.....	4
Détermination de l'algorithme de Monte Carlo pour π	5
L'algorithme en code.....	6
Parallélisation - Paradigme Master Worker.....	6
Evaluation des performances (PC G26).....	7
Evaluation des performances (Mac M3).....	8
Distribution - MasterSocket / WorkerSocket.....	9
Qualité de développement.....	10
Notes pas à pas.....	10
Important / À retenir.....	11

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.

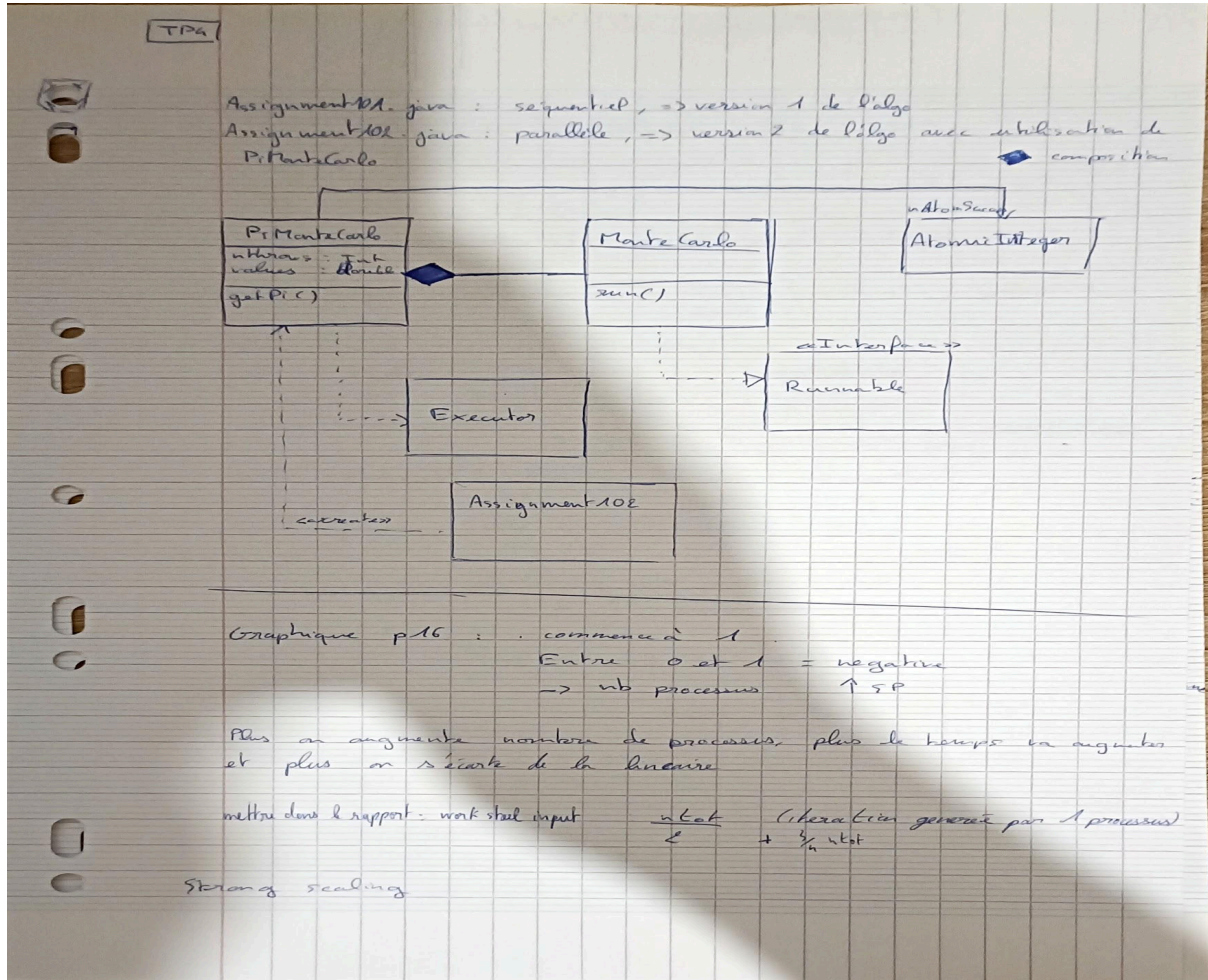
La scalabilité forte permet d'évaluer si l'on va plus vite avec plus de processus = Question de temps.

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 Assignment102

Assignment102 (Première version de PI avec MonteCarlo)



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.

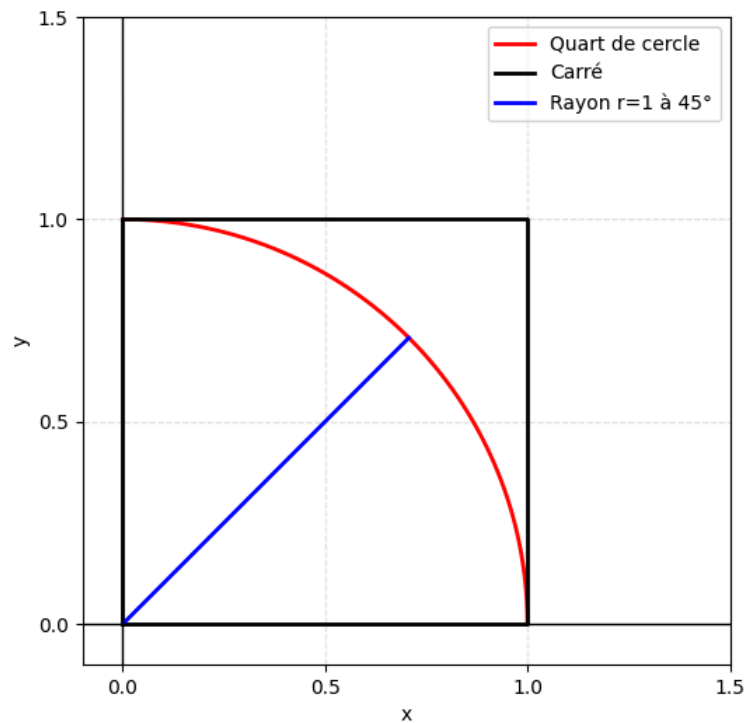
Exemples :

- 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 $A_c = r^2 = 1 \cdot 1 = 1$
- L'aire du quart de disque s'écrit $A_{d4} = \pi \cdot r^2 / 4 = \pi/4$
- On effectue n_{total} tirages

En suite, on sait que la probabilité qu'un point X_i soit dans le quart de disque si sa valeur d_i est comprise dans le quart de disque :

$$P(X_i \mid d_i < 1) = A_{d4} / A_c = (\pi/4) / 1 = \pi/4$$

En sachant que chaque point X_i a deux coordonnées x_i et y_i et que sa valeur d_i est obtenue par $d_i = \sqrt{x_i^2 + y_i^2}$

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 X_i soit dans le quart de disque est :

$$P(X_i \mid d_i < 1) = n_{\text{cibles}} / n_{\text{total}}$$

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 :

$$\begin{aligned} \pi/4 &= n_{\text{cibles}} / n_{\text{total}} \\ \Leftrightarrow \pi &= 4 * n_{\text{cibles}} / n_{\text{total}} \end{aligned}$$

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

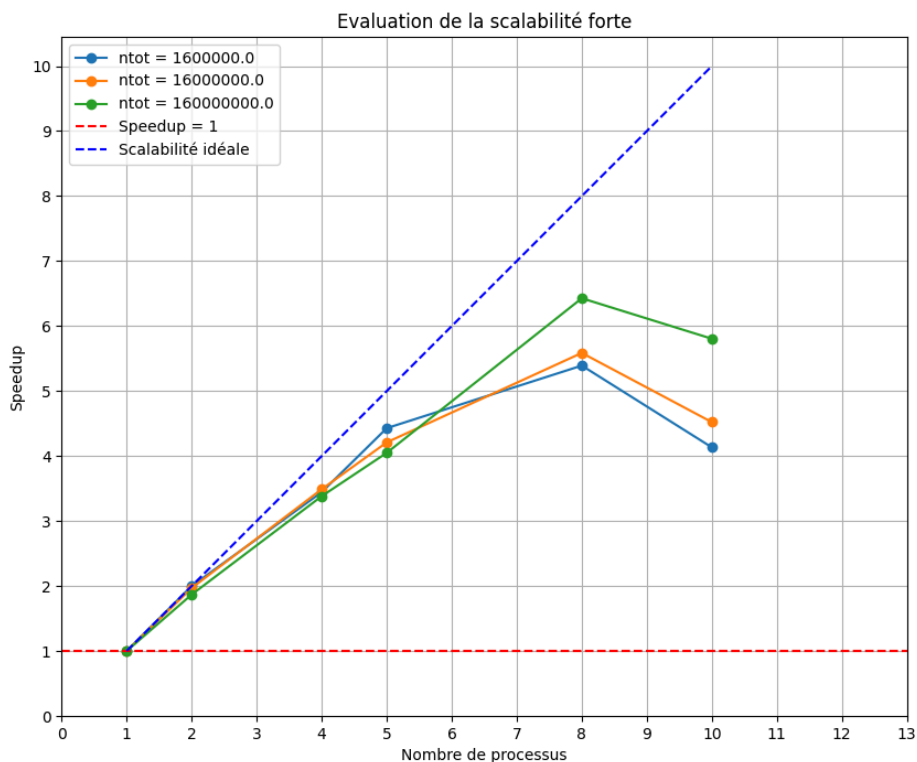
Evaluation des performances (PC G26)

Sur PC : Le processeur Intel Core I7-7700 possède 4 coeurs physiques hyper threadé à 8.

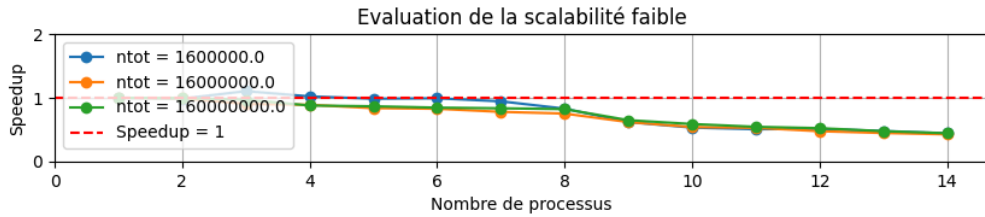
Dans le graphique de scalabilité forte ci-dessous, on observe une performance assez proche de l'idéal jusqu'à 5-6 processus. Ensuite, cela se stabilise jusqu'à 8 (Le maximum) jusqu'à s'éloigner totalement car au-delà de 8 processus, les processus suivants utilisent les mêmes ressources que les premiers.

On observe également que plus la taille de l'expérience est élevée (ntot), meilleure est la scalabilité.

On en déduit qu'une plus grande charge de travail permet de mieux exploiter la parallélisation.



Dans le graphique de scalabilité faible ci-dessous, on observe que jusqu'à 8 processus, les courbes restent assez proches de 1 puis se rapprochent de 0 au-delà. On peut en déduire que l'augmentation proportionnelle de la taille du problème par rapport au nombre de processus ne dégrade pas les performances à part au delà de 8 processus, ce qui est normal.



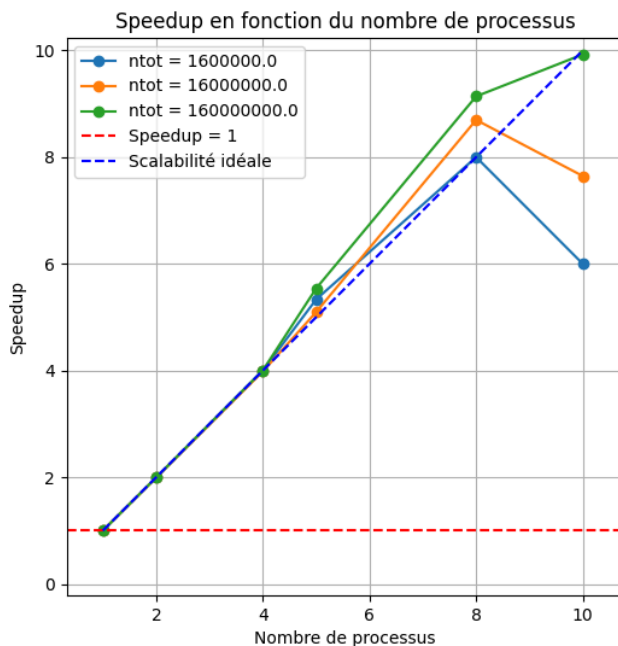
Evaluation des performances (Mac M3)

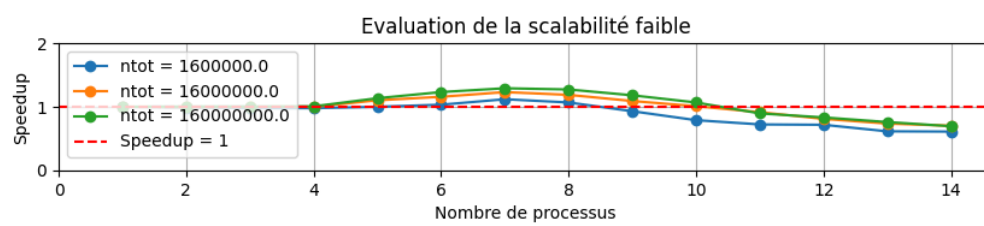
Sur puce M3 (8 coeurs CPU 10 coeurs GPU) (4 coeurs de performance, 4 coeurs d'efficacité)

L'exécution du même code sur mon mac montre des résultats inattendus pour l'évaluation de la scalabilité forte et faible.

En effet, dans le premier graphique, jusqu'à 8 processus, j'observe des courbes au-dessus de la scalabilité idéale ce qui voudrait dire que je ne tire pas partie de la parallélisation.

Dans le deuxième également, les courbes sont proches de 1 puis au-dessus de 1 et finalement en-dessous.





Distribution - MasterSocket / WorkerSocket

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 intéressés par ça mais plutôt les résultats (Ex : Il y a un bug => à quelle vitesse je peux trouver les bugs et les corriger 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]

Modularity : 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

Effectiveness (efficacité) : Exactitude avec laquelle les utilisateurs atteignent les objectifs spécifiés

Efficiency (efficience) : Exactitude avec laquelle les utilisateurs atteignent les objectifs spécifiés