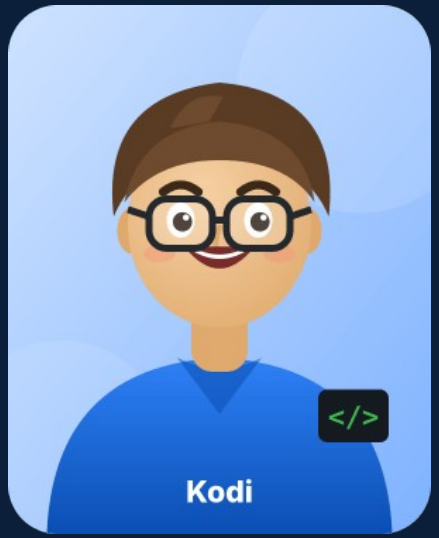
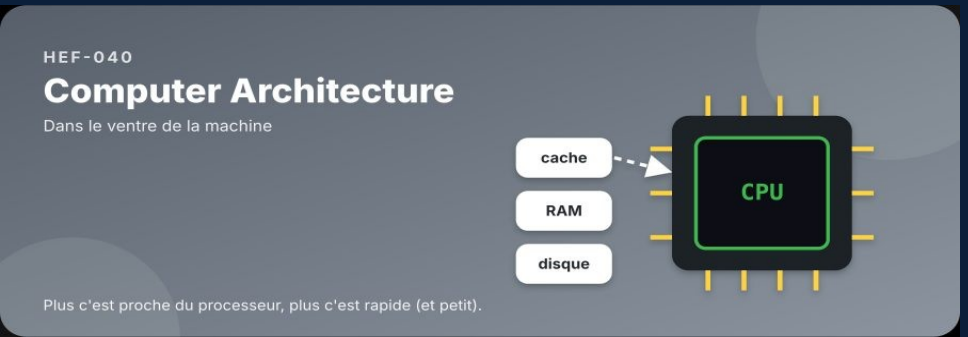
HEF-040 · MISSION 09 / 10

Performance et limites physiques

Pourquoi les processeurs n'ont pas une fréquence de 100 GHz ? Parce que la physique répond non.





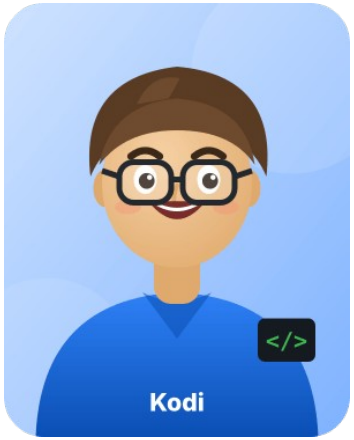
Badge : Physicien·ne du silicium

75 min · débutant

L'histoire

Maya demande : « On achète une machine deux fois plus rapide, et notre rapport ira deux fois plus vite, non ? » Kodi hésite. L'ancien serveur avait 4 cœurs à 3 GHz, le nouveau en a 16 à 2,5 GHz.

> *« Ça dépend de ton programme, dit Sam. S'il ne sait utiliser qu'un seul cœur, ta machine sera plus lente. Et il y a des limites que l'argent ne peut pas acheter. »*



Tu comprends ce qui fait (et limite) la performance d'un processeur.

Ta mission

À la fin, je serai capable de…

1. relier fréquence, IPC et performance
2. expliquer le pipeline et le coût d'une erreur de prédiction de branchement
3. expliquer le parallélisme (cœurs, threads) et la loi d'Amdahl
4. citer les limites physiques : chaleur, vitesse de la lumière, mémoire
5. choisir entre « plus rapide » et « plus de cœurs » selon la charge

Défi d'échauffement

Défi d'échauffement

Dans une usine à sandwiches : une personne fait tout (pain, garniture, emballage) en 3 minutes par sandwich. Si tu embauches 3 personnes, chacune spécialisée dans une étape, combien de sandwiches par heure en régime établi ? Combien de temps pour le premier sandwich ? Que se passe-t-il si une étape est plus lente que les autres ?

Le concept

La formule de base

temps d'exécution = nombre d'instructions × cycles par instruction ÷ fréquence

- **Fréquence** (GHz) : cycles par seconde.
- **IPC** (*instructions per cycle*) : nombre d'instructions terminées par cycle. Les processeurs modernes en terminent plusieurs par cycle.

Fréquence plus haute **ou** IPC plus grand **ou** moins d'instructions → programme plus rapide.

Le pipeline

Le concept

Le pipeline (suite)

Au lieu d'exécuter une instruction en entier avant de commencer la suivante, le CPU la découpe en étapes (chercher, décoder, exécuter, écrire) et fait travailler les étapes en parallèle sur des instructions différentes, comme une chaîne de montage.

```
Sans pipeline : 4 instructions x 4 étapes = 16 cycles
Avec pipeline : 4 + (4 - 1) = 7 cycles
```

Avec k étapes et n instructions : **n + k − 1** cycles au lieu de **n × k**.

Les aléas : quand le pipeline se bloque

- **Branchements** (`if`, boucles) : le CPU ne sait pas quelle instruction vient ensuite avant d'avoir évalué la condition. Il **prédit** et continue. Si la prédiction est fausse, il **vide** le pipeline : perte de ~10 à 20 cycles.
- **Dépendances de données** : une instruction a besoin du résultat de la précédente.

Le concept

Les aléas : quand le pipeline se bloque (suite)

- **Attente mémoire** : un échec de cache bloque tout (mission 5).

Le parallélisme

Niveau	Idée
Pipeline / superscalaire	plusieurs instructions en même temps dans un cœur
Multicœur	plusieurs cœurs, chacun exécute son fil d'instructions
SIMD	une instruction traite plusieurs données
GPU	des milliers de petits cœurs pour des calculs parallèles

La loi d'Amdahl

Si une fraction **p** du travail peut être parallélisée sur **N** cœurs, l'accélération maximale est :

Le concept

La loi d'Amdahl (suite)

accélération = 1 / ((1 − p) + p / N)

```
def amdahl(p, n):
    return 1 / ((1 - p) + p / n)

for p in (0.5, 0.9, 0.99):
    print(p, [round(amdahl(p, n), 1) for n in (2, 4, 16, 1024)])
```

Avec 50 % de parallélisable, 1 024 cœurs ne feront jamais mieux que **2 fois** plus vite.

Les murs physiques

Mur	Explication
Chaleur (power wall)	la puissance dissipée croît vite avec la fréquence : au-delà de ~5 GHz, refroidir devient impraticable

Le concept

Les murs physiques (suite)

Mur	Explication
Vitesse de la lumière	en 1 ns, un signal parcourt au mieux ~30 cm dans le vide, moins dans le cuivre ; à 5 GHz, un cycle dure 0,2 ns
Mur mémoire	le CPU progresse plus vite que la mémoire : la latence de la RAM stagne (mission 4)
Miniaturisation	les transistors approchent la taille de quelques atomes

C'est pourquoi, depuis le milieu des années 2000, on ajoute des **cœurs** plutôt que d'augmenter la fréquence.

🧠 ***Astuce** : un programme séquentiel ne profite pas de 16 cœurs. Le parallélisme doit être **dans le programme**.*

Exemple pas à pas

Problème : comparer deux machines pour un traitement dont 80 % est parallélisable.

- A : 4 cœurs à 3,5 GHz ;
- B : 16 cœurs à 2,5 GHz.

On suppose le même IPC. Temps relatif pour un travail qui prendrait 1 unité sur un cœur à 1 GHz :

```
def temps(p, coeurs, ghz):
    return ((1 - p) + p / coeurs) / ghz

for p in (0.8, 0.5, 0.99):
    a = temps(p, 4, 3.5)
    b = temps(p, 16, 2.5)
    meilleur = "A" if a < b else "B"
    print(f"parallélisable {p:4.0%} : A={a:.3f}  B={b:.3f}  → {meilleur} gagne")
```

Exemple pas à pas (suite)

Lecture : à 50 % de parallélisme, la machine A (plus rapide par cœur) gagne ; dès 80 %, la machine B l'emporte, et l'écart se creuse à 99 %. Le bon choix dépend de la **charge**, pas du nombre brut de cœurs ou de GHz.

Pipeline : coût d'un branchement mal prédit

```
def cycles_pipeline(n_instructions, etapes=5, branchements=0, penalite=15, taux_er
   reur=0.0):
    base = n_instructions + etapes - 1
    return base + branchements * taux_erreur * penalite

print(cycles_pipeline(1000, branchements=200, taux_erreur=0.02))   # prédicteur ex
   cellent
print(cycles_pipeline(1000, branchements=200, taux_erreur=0.30))   # données impré
   visibles
```

À toi, avec aide

Une tâche prend 10 secondes sur un cœur. 90 % est parallélisable.

1. Calcule le temps avec 2, 4, 8, 16 et 1 000 cœurs (loi d'Amdahl).
2. Quelle est la limite quand le nombre de cœurs tend vers l'infini ?
3. Que faudrait-il faire pour aller encore plus vite ?

À toi, avec aide · réponse

Indices progressifs

Indice 1

Temps = 10 × ((1 − p) + p / N).

Indice 2

La partie non parallélisable vaut 10 % de 10 s = 1 s.

Indice 3

Réduire la partie séquentielle (changer d'algorithme, de structure).

À toi, avec aide · réponse

Réponse

```
def temps(base, p, n):
    return base * ((1 - p) + p / n)

for n in (1, 2, 4, 8, 16, 1000):
    print(n, round(temps(10, 0.9, n), 2), "s")
# 10.0, 5.5, 3.25, 2.12, 1.56, 1.01
```

La limite est 1 seconde (la partie séquentielle). Pour descendre en dessous, il faut réduire la partie séquentielle elle-même : meilleur algorithme, moins de synchronisation, découpage différent du problème.

À toi, en solo

Simule un **prédicteur de branchement** simple. On te donne l'historique d'un branchement (`True` = pris, `False` = non pris) :

1. Prédicteur « toujours pris » ;
2. Prédicteur « comme la dernière fois » ;
3. Prédicteur à compteur de 2 bits (saturant de 0 à 3 : prédit « pris » si ≥ 2).

Compare leur taux d'erreur sur : (a) une boucle de 100 itérations (99 pris, 1 non pris, répété 10 fois) ; (b) un branchement aléatoire 50/50 ; (c) un motif alterné `True, False, True, False…`.

À toi, en solo · réponse

Pistes de correction

```
import random

def toujours_pris(hist):
    return sum(1 for h in hist if not h)

def derniere_fois(hist):
    erreurs, prec = 0, True
    for h in hist:
        erreurs += (prec != h)
        prec = h
    return erreurs

def deux_bits(hist):
    compteur, erreurs = 2, 0
    for h in hist:
        prediction = compteur >= 2
        erreurs += (prediction != h)
        compteur = min(3, compteur + 1) if h else max(0, compteur - 1)
    return erreurs
```

À toi, en solo · réponse

Pistes de correction (suite)

```
boucle = ([True] * 99 + [False]) * 10
rng = random.Random(1)
aleatoire = [rng.random() < 0.5 for _ in range(1000)]
alterne = [i % 2 == 0 for i in range(1000)]

for nom, hist in (("boucle", boucle), ("aléatoire", aleatoire), ("alterné", altern
   e)):
    n = len(hist)
    print(nom, [f"{100 * f(hist) / n:.1f}%" for f in (toujours_pris, derniere_fois
   , deux_bits)])
```

Sur la boucle, « toujours pris » et le compteur de 2 bits se trompent environ une fois sur cent (seulement à la sortie de boucle) ; « comme la dernière fois » se trompe deux fois par sortie (1,9 %). L'aléatoire reste autour de 50 % quoi qu'on fasse : aucune prédiction ne peut deviner. L'alterné piège « comme la dernière fois » (près de 100 % d'erreurs), alors que le compteur de 2 bits reste à 50 %.

Pièges classiques

Piège	Pourquoi c'est un problème	Comment l'éviter
Croire que plus de GHz = plus rapide	L'IPC et la mémoire comptent aussi	Comparer sur la charge réelle
Croire que plus de cœurs accélère tout	Un programme séquentiel n'en profite pas	Appliquer la loi d'Amdahl
Ignorer les branchements imprévisibles	Pipeline vidé à chaque erreur	Rendre les données/branches prévisibles quand c'est critique
Négliger le mur mémoire	Le CPU attend la RAM	Améliorer la localité d'abord

Mini-boss

Choisir un serveur pour HashBox. Trois charges de travail :

1. **Serveur web** : des milliers de requêtes indépendantes, 98 % parallélisable.
2. **Rapport mensuel** : un seul gros calcul, 30 % parallélisable.
3. **Analyse de données** : 90 % parallélisable, mais très dépendante de la mémoire.

Candidats : A (8 cœurs, 3,8 GHz), B (32 cœurs, 2,4 GHz), C (16 cœurs, 3,0 GHz, mémoire deux fois plus rapide).

1. Écris un programme qui calcule le temps relatif de chaque machine pour les charges 1 et 2 avec la loi d'Amdahl.
2. Pour la charge 3, propose un modèle simple (par exemple multiplier le temps par un facteur selon la vitesse mémoire) et justifie-le.
3. Fais une recommandation pour une machine unique polyvalente.

Mini-boss · réponse

Pistes de correction

```
def temps(p, coeurs, ghz):
    return ((1 - p) + p / coeurs) / ghz

machines = {"A": (8, 3.8, 1.0), "B": (32, 2.4, 1.0), "C": (16, 3.0, 2.0)}   # (coe
   urs, GHz, facteur mémoire)
charges = {"serveur web": 0.98, "rapport mensuel": 0.30, "analyse": 0.90}

for nom_c, p in charges.items():
    resultats = {}
    for nom_m, (c, g, mem) in machines.items():
        t = temps(p, c, g)
        if nom_c == "analyse":
            t = t * (0.5 + 0.5 / mem)       # moitié du temps dépend de la mémoire
        resultats[nom_m] = round(t, 4)
    print(nom_c, resultats, "→ meilleure :", min(resultats, key=resultats.get))
```

Mini-boss · réponse

Pistes de correction (suite)

Attendu : B pour le serveur web, A pour le rapport mensuel (peu parallélisable : la fréquence compte), C pour l'analyse (mémoire plus rapide). Si une seule machine doit être polyvalente, C est un bon compromis ; la recommandation doit citer les compromis et l'incertitude du modèle.

Quiz éclair

Quiz éclair

1. La loi d'Amdahl dit que l'accélération par parallélisme est limitée par :

A. la fréquence
B. la partie non parallélisable
C. la taille de la RAM

2. Un pipeline permet de :

A. augmenter la taille du cache
B. réduire la RAM
C. traiter plusieurs instructions en même temps à des étapes différentes

3. Une mauvaise prédiction de branchement :

A. vide le pipeline et fait perdre des cycles

Quiz éclair (suite)

B. accélère le programme
C. corrompt la mémoire

4. 🔁 *Révision mission 08* : Un cadre de pile contient notamment :

A. le code source
B. les variables locales et l'adresse de retour
C. les fichiers ouverts

5. 🔁 *Révision mission 07* : Le DMA permet :

A. de chiffrer des fichiers
B. d'augmenter la fréquence
C. de copier des données sans mobiliser le CPU

Quiz éclair · réponse

Réponses

1. **B**.
2. **C**.
3. **A**.
4. **B**.
5. **C**.

Je réfléchis

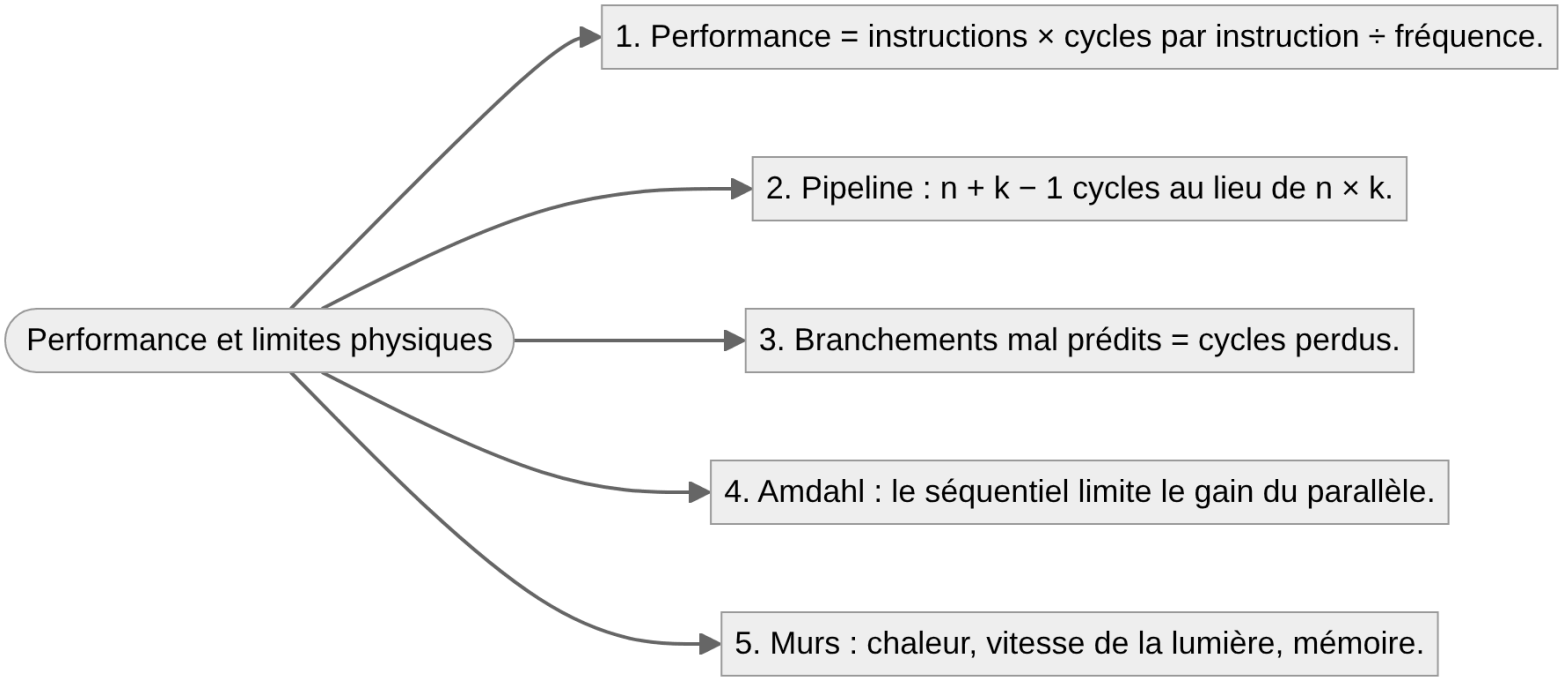
1. Dans quelle situation de ta vie « ajouter des personnes » n'accélère-t-il pas un projet ?
2. Qu'est-ce qui te frappe le plus parmi les limites physiques des processeurs ?

À retenir

1. Performance = instructions × cycles par instruction ÷ fréquence.
2. Pipeline : n + k − 1 cycles au lieu de n × k.
3. Branchements mal prédits = cycles perdus.
4. Amdahl : le séquentiel limite le gain du parallèle.
5. Murs : chaleur, vitesse de la lumière, mémoire.

À retenir

Le schéma de la mission



MISSION ACCOMPLIE

+100 XP

Badge débloqué : Physicien·ne du silicium

Prochaine mission : Étude d'une configuration réelle

