Programmation concurrente

Avantages et pièges

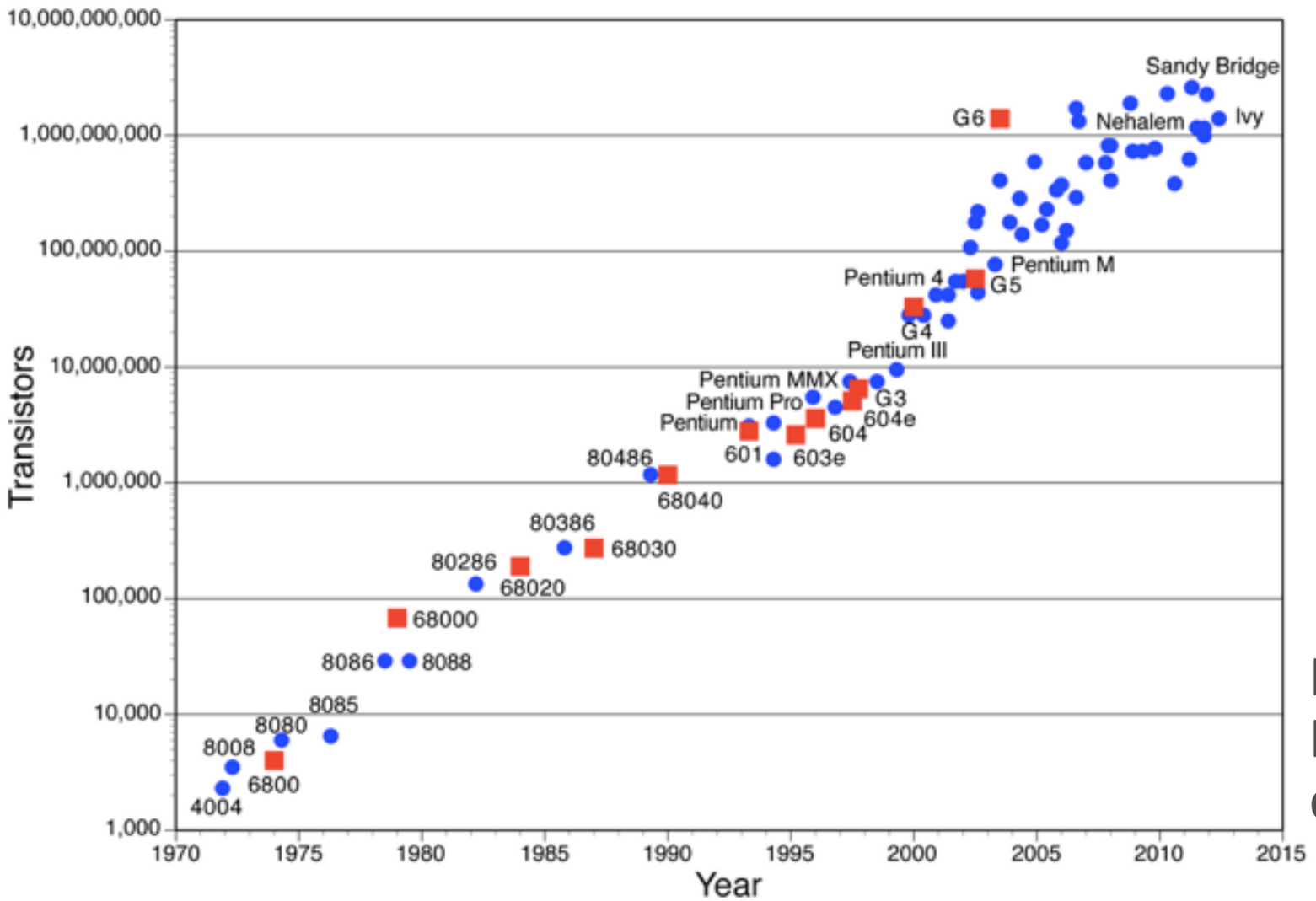
Programmation concurrente

- Avantages de la programmation concurrente
  - Amélioration des performances
    - Maximisation de l'utilisation du processeur
      - Utilisation du processeur pendant les phases d'attente
        - Une lecture sur le disque ou un accès réseau durent une éternité à l'échelle de temps du processeur ➙ mise de côté du thread pour faire avancer le reste du travail
    - Mise à profit des architectures multicoeurs, multiprocesseurs (➙ utilisation des processeurs de la carte graphique (OpenCL, Cuda), calcul distribué)
  - Permet de "modulariser" une application (découpage en tâches élémentaires) ➙ simplification du code
    - Surveillance de capteurs avec des périodicités différentes

Programmation concurrente

Avantages de la programmation concurrente

Amélioration des performances

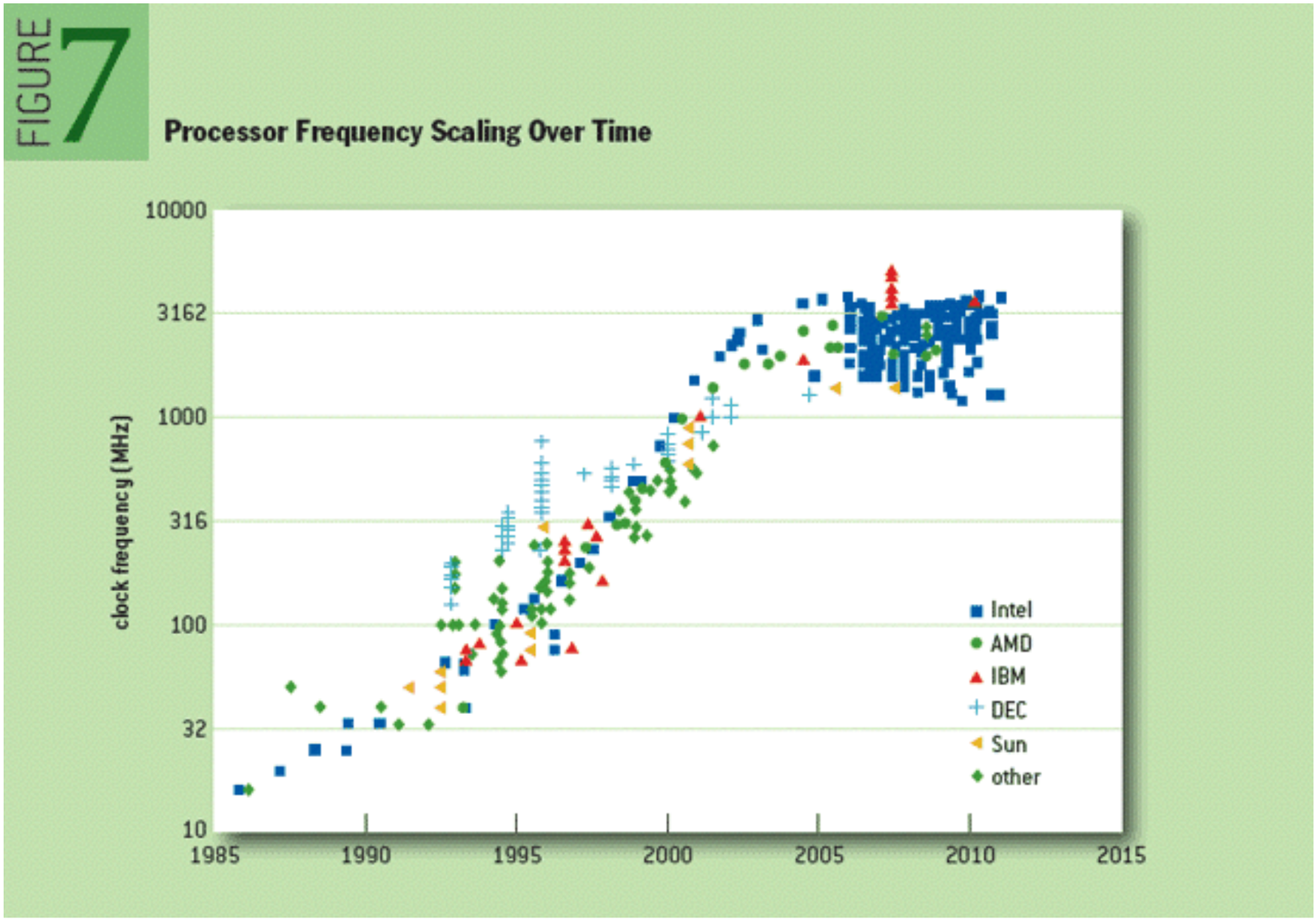


Loi de Moore
Le nombre de transistors double tous les 18 mois

http://theinnovationandstrategyblog.com/2014/07/linnovation-et-le-progres-est-ce-la-meme-chose/

Programmation concurrente

- Avantages de la programmation concurrente
  - Amélioration des performances



La fréquence des coeurs plafonne ➙ multi-coeurs

de 2 à 4 coeurs dans les mobiles d'aujourd'hui (les processeurs mobiles 8 coeurs sont déjà prêts)

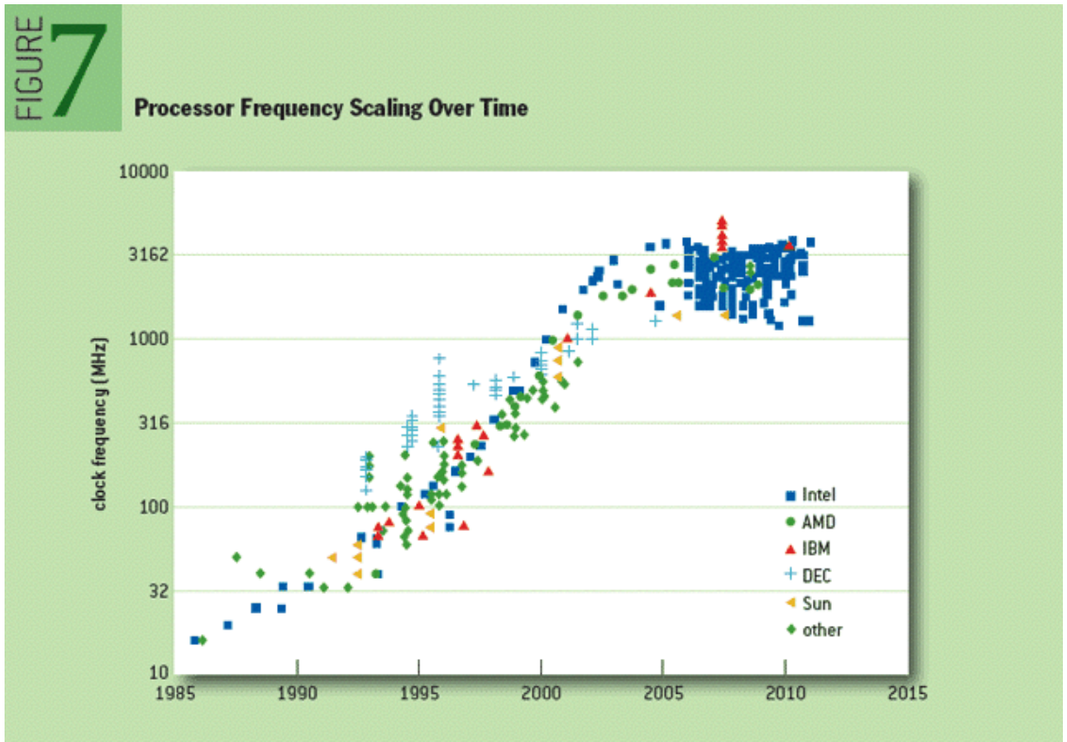
512 coeurs d'ici 10 ans ?

https://queue.acm.org/detail.cfm?id=2181798

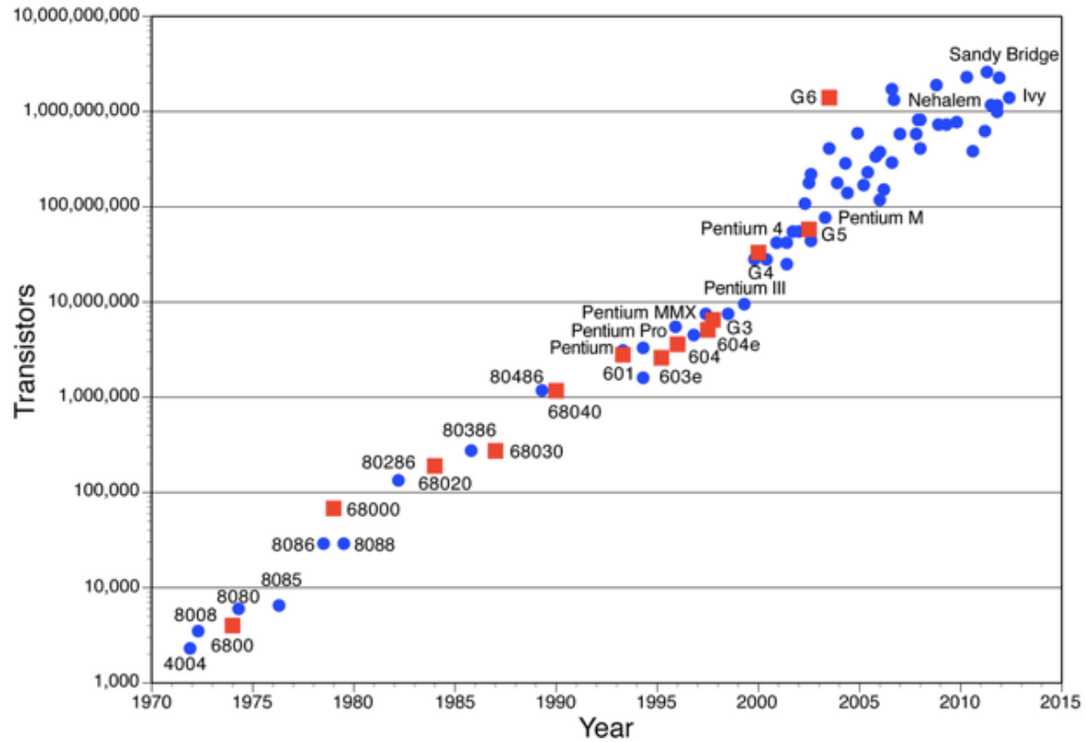
Programmation concurrente

Avantages de la programmation concurrente

Amélioration des performances



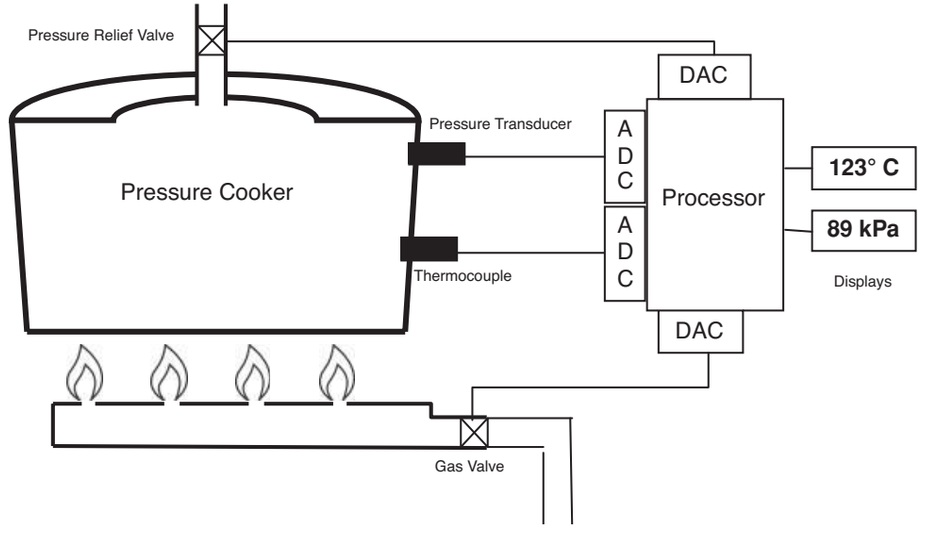
+



Nécessité pour les développeurs de maîtriser la programmation concurrente

Programmation concurrente

- Avantages de la programmation concurrente
  - Exemple : cuisson avec régulation de température et pression - Solution avec un seul thread
    - Dans un premier temps on choisit une période de 5 s pour les deux régulations



loop
  Convertir la pression (ADC)
  Calculer la commande de la vanne d'évacuation
  Sortir la commande de la vanne (DAC)

  Lire la température (ADC)
  Calculer la commande de la vanne de gaz
    Sortir la commande de la vanne (DAC)

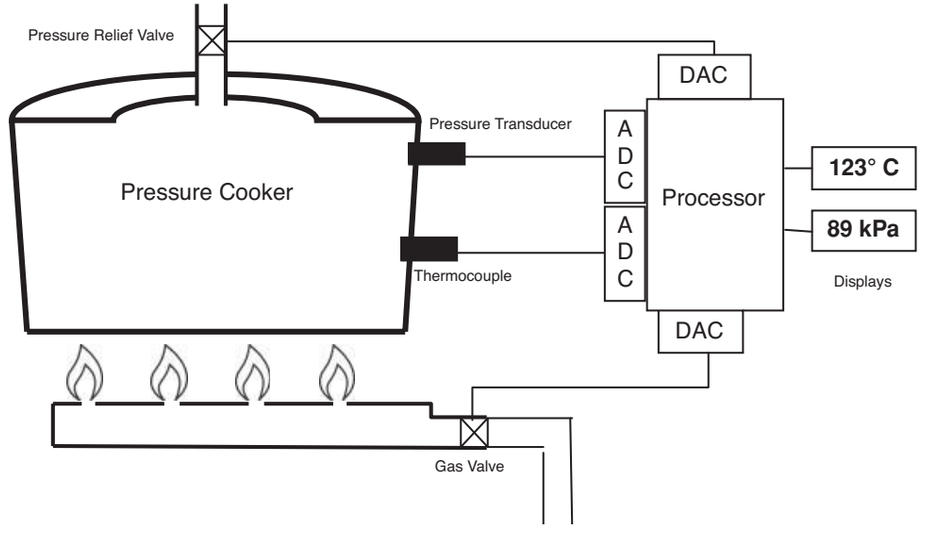
  Attendre que 5 s se soient écoulée depuis le début de la boucle
end loop

Programmation concurrente

- Avantages de la programmation concurrente
  - La pression fluctue trop, un expert en régulation suggère de diminuer la période à 2 s pour la pression et à 4 s pour la T°

loop
  Convertir la pression (ADC)
  Calculer la commande de la vanne d'évacuation
  Sortir la commande de la vanne (DAC)
  Lire la température (ADC)
  Calculer la commande de la vanne de gaz
    Sortir la commande de la vanne (DAC)
  Attendre que 2 s se soient écoulée depuis le début de la boucle
    Convertir la pression (ADC)
    Calculer la commande de la vanne d'évacuation
    Sortir la commande de la vanne (DAC)
    Attendre que 2 s se soient écoulée depuis la dernière conversion de pression

end loop



Programmation concurrente

Avantages de la programmation concurrente

- Finalement cette solution ne fait pas l'affaire et on opte pour une période de 1,5 s pour la pression et de 2,5 s pour la T°

loop
    Conversion/Calcul/Commande pression
    Conversion/Calcul/Commande temperature
    Attendre que 1,5 s se soient écoulée depuis le début de la boucle
    Conversion/Calcul/Commande pression
    Attendre que 1 s se soient écoulée depuis la conversion précédente
    Conversion/Calcul/Commande temperature
    Attendre que 0,5 s se soient écoulée depuis la conversion précédente
    Conversion/Calcul/Commande pression
    Attendre que 1,5 s se soient écoulée depuis la conversion précédente
    Conversion/Calcul/Commande pression
    Attendre que 0,5 s se soient écoulée depuis la conversion précédente
    Conversion/Calcul/Commande temperature
    Attendre que 1 s se soient écoulée depuis la conversion précédente
    Conversion/Calcul/Commande pression
    Attendre que 1,5 s se soient écoulée depuis la conversion précédente
end loop

Programmation concurrente

- Avantages de la programmation concurrente
  - Gestion de périodicité multiple
    - Solution beaucoup plus souple et simple avec 2 threads

Thread Regulation Pression
loop
    Conversion/Calcul/Commande pression
    Attendre que 1,5 s se soient écoulée depuis le début de la boucle
end loop

Thread Regulation Temperature
loop
    Conversion/Calcul/Commande temperature
    Attendre que 2,5 s se soient écoulée depuis le début de la boucle
end loop

Programmation concurrente

- Avantages de la programmation concurrente
  - Interface utilisateur plus réactive
  - Réalisation d'opérations en tâche de fond :
    - Recherche réseau de mises à jours
    - Vérification d'orthographe
    - Téléchargements de nouveaux articles (fils RSS, applications de news,...)

Programmation concurrente

- Avantages de la programmation concurrente
  - Démarrage d'application rapide ➙ fondamental pour une application mobile. 2 stratégies possibles :
    - Lazy loading : on retarde l'instanciation des objets, voire l'initialisation des champs et le chargement des ressources jusqu'au moment où en a réellement besoin
      - Pas de temps perdu à générer des éléments qui ne seront pas utilisés
    - Chargement dans un thread d'arrière plan
      - Si les ressources sont longues à charger, on peut anticiper les besoins de l'utilisateur
      - Chargement du menu de départ puis chargement en tâche de fond des ressources nécessaires (modèles 3D, données, ...)

Programmation concurrente

- Avantages de la programmation concurrente
  - Amélioration des performances par parallélisation d'algorithmes gourmands
    - Traitement de l'image
    - Intelligence artificielle : réseaux de neurones, logique floue, algorithmes évolutionnistes (génétiques), bases de règles, jeux de stratégie
    - Recherche de solutions (Optimisation, calcul scientifique, recherche de chemin)
    - Web crawler
    - Indexation système de fichiers

Programmation concurrente

- Risques de la programmation concurrente
  - Risques liés au partage de données entre tâches
    - accès concurrents
  - Problèmes si la validité du résultat dépend d'un timing chanceux
    - ces problèmes ne se manifestent pas systématiquement
      - potentiellement invisibles lors des tests
      - difficiles à reproduire

Ce n'est pas parce que le programme passe les tests qu'il "marche" !

Programmation concurrente

- Risques de la programmation concurrente

“ Concurrent programming is **REALLY** hard ! ”

Beginner

“ Concurrent programming is not that hard ! ”

Intermediate programmer

“ Concurrent programming is **REALLY** hard ! ”

Expert programmer

Risques de la programmation concurrente

“ Everybody who learns concurrency thinks they understand it, ends up finding mysterious races they thought weren’t possible, and discovers that they didn’t actually understand it yet after all. ”

Herb Sutter, chair of the ISO C++ standards committee, Microsoft.

“ If POSIX threads are a good thing, perhaps I don’t want to know what they’re better than. ”

Rob Pike,Ingénieur chez Google (un des créateurs du langage Go)

Programmation concurrente

- Risques de la programmation concurrente
  - Les failles de concurrence portent le nom de *race conditions* et elles impliquent des séquences de type :
    - *Check-then-act*
    - *Read-Modify-Write*

Programmation concurrente

- *Race conditions* - exemples
  - Compte bancaire (Bruno & Sylvie)
    - Check-then-act
      - Solde du compte 100 €
        - Bruno se connecte sur le site de la banque pour vérifier le solde et il en déduit qu'il peut effectuer un achat de 80 €
        - Avant qu'il ait terminé son achat en ligne, Sylvie se connecte également sur le site de la banque et vérifie que le solde de 100 € est suffisant pour son achat de 60 €
        - Cette situation conduit à un découvert de 40 €

Programmation concurrente

- *Race conditions* - exemples
  - Compte bancaire (Bruno & Sylvie) - Mise à jour perdue
    - Read-Modify-Write
      - Solde du compte 100 €
      - Le système informatique de la banque reçoit la demande de virement du salaire de Sylvie (2000 €) et lit le solde du compte (100 €)
      - Une deuxième tâche reçoit la demande de virement du salaire de Bruno (2000 €) et lit le solde qui est toujours de 100 €
      - La première tâche écrit la valeur du solde qu'elle a calculée (2100 €)
      - La deuxième tâche écrit la valeur du solde qu'elle a calculée (2100 €)
      - 2000 € ont disparu !

Programmation concurrente

- *Race conditions*
  - *Check-then-act*
  - *Read-Modify-Write*
- Solution : atomicité
  - Le système doit fournir des services permettant de garantir que la séquence complète pourra se dérouler sans conflit
  - Les séquences identifiées doivent faire l'objet d'une exclusion mutuelle

- *Race conditions*
  - Autre exemple plus subtil de partage sans séquence Read-Modify-Write ni Check-Then-Act :
    - Les accès aux variables long en Java ne sont pas garantis d'être atomiques (c'est également le cas pour les processeurs dont la taille du coeur est inférieure à la taille des variables : processeur 32 bits avec variables sur 64 bits)
  - Si un thread incrémente un long qui valait -1
    - En complément à 2, -1 est représenté par 0xFFFFFFFF FFFFFFFF
    - On doit remplacer cette valeur par 0x00000000 00000000
    - Cela peut se faire en 2 écritures de 32 bits
    - Un thread peut observer la variable dans un état intermédiaire
      - 0x00000000 FFFFFFFF (4294967295)
      - 0xFFFFFFFF 00000000 (-4294967296)

Programmation concurrente

- Résolution des problèmes de partage
  - Exclusion mutuelle
    - Instructions atomiques du processeur
      - Compare And Swap (CAS), Test And Set (TAS)
    - Semaphores, mutex, verrous (locks), ...
      - ces outils permettent limiter l'accès à des séquences critiques
      - Il est nécessaire de s'appuyer sur le même objet de synchronisation (semaphore, lock,...) à **tous** les endroits du code qui accèdent à la séquence critique
        - En conception orientée objet une bonne encapsulation peut aider à garantir cette condition

Programmation concurrente

- Interblocage
  - La mise en attente d'une tâche lorsqu'une ressource est indisponible peut conduire à une situation d'interblocage (*deadlock*)
  - L'interblocage apparaît lorsque plusieurs mutex, verrous,... doivent être acquis et que l'acquisition par les threads se fait dans un ordre différent

```
Thread A

leftMotorSemaphore.acquire()
rightMotorSemaphore.acquire()
//Commande du mouvement
...
```

```
Thread B

rightMotorSemaphore.acquire()
leftMotorSemaphore.acquire()
//Commande du mouvement
...
```

Programmation concurrente

- Interblocage

```
Thread A
leftMotorSemaphore.acquire()
```

```
rightMotorSemaphore.acquire()
```

```
Thread B
rightMotorSemaphore.acquire()
```



```
leftMotorSemaphore.acquire()
```

Les 2 threads s’attendent mutuellement