# Conception de circuits numériques et architecture des ordinateurs

Frédéric Pétrot

GRENOBLE INP UGA Ensimag

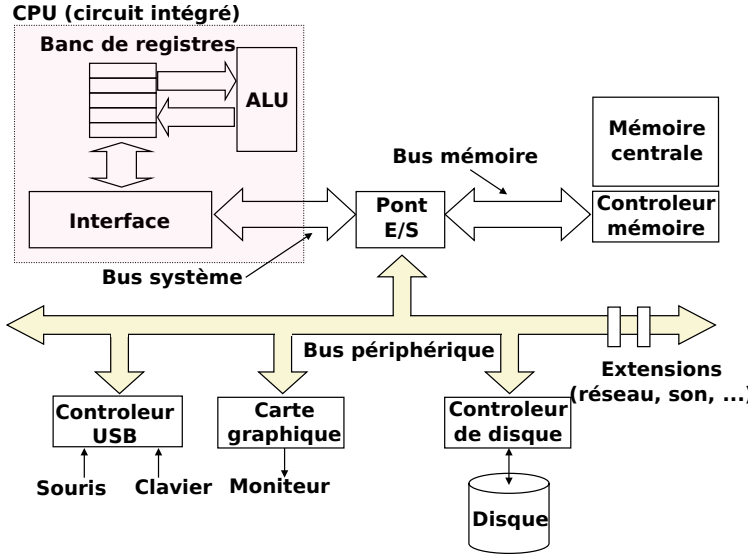
Année universitaire 2022-2023

| | |
|---|---|
| C1 | Codage des nombres en base 2, logique booléenne, portes logiques, circuits combinatoires |
| C2 | Circuits séquentiels |
| C3 | Construction de circuits complexes |
| C4 | Micro-architecture et fonctionnement des mémoires |
| C5 | Machines à état |
| C6 | Synthèse de circuits PC/PO |
| C7 | Optimisation de circuits PC/PO |
| C8 | Interprétation d'instructions - 1 |
| C9 | Interprétation d'instructions - 2 |
| C10 | Interprétation d'instructions - 3 |
| C11 | Introduction aux caches |

# Plan détaillé du cours d'aujourd'hui

# Plan

# Structure d'une machine



# Structure d'une machine

## Accès mémoire centrale

Processeur émet requête activant :
Interface processeur → bus système → pont → bus mémoire → contrôleur mémoire → mémoire

- Purement électronique, symétrique
- Processeur en attente du résultat

## Accès disque

Processeur émet (commande, numéro de secteur, adresse) :
Interface E/S → bus système → pont → bus périphérique → contrôleur de disque
Contrôleur disque → disque → bus périphérique → pont → bus mémoire → contrôleur mémoire → mémoire (DMA)
Contrôleur disque → processeur (interruption) → accès lecture mémoire

- Électronique et mécanique
- Transfert asymétrique
- Processeur passe à une autre tâche (système d'exploitation)

# À propos du stockage des données

Évolution de la taille et du temps d'accès (données approximatives, ordres de grandeur)

## Mémoire statique

| Metrique | 1980 | 1985 | 1990 | 1995 | 2000 | 2005 | 2010 | 2015 | 2020 | 2020 vs 1980 |
|---|---|---|---|---|---|---|---|---|---|---|
| $/MB | 19,200 | 2,900 | 320 | 256 | 100 | 75 | 30 | 10 | 5 | ÷3840 |
| $T_{acc}$ (ns) | 300 | 150 | 35 | 15 | 3 | 2 | 1.5 | 1.3 | 1 | ÷300 |

## Mémoire dynamique

| Metrique | 1980 | 1985 | 1990 | 1995 | 2000 | 2005 | 2010 | 2015 | 2020 | 2020 vs 1980 |
|---|---|---|---|---|---|---|---|---|---|---|
| $/MB | 8,800 | 880 | 100 | 30 | 1 | 0.1 | 0.02 | 0.005 | 0.003 | ÷2,930,000 |
| $T_{acc}$ (ns) | 375 | 200 | 100 | 70 | 60 | 50 | 40 | 30 | 30 | ÷12.5 |
| Taille typ.(MB) | 0.064 | 0.256 | 4 | 16 | 64 | 2,000 | 8,000 | 16,000 | 32,000 | ×500,000 |

## Disque

| Metrique | 1980 | 1985 | 1990 | 1995 | 2000 | 2005 | 2010 | 2015 | 2020 | 2020 vs 1980 |
|---|---|---|---|---|---|---|---|---|---|---|
| $/MB | 500 | 100 | 8 | 0.30 | 0.01 | 0.005 | 0.0003 | 0.00003 | .000018 | ÷27,800,000 |
| $T_{acc}$ (ms) | 87 | 75 | 28 | 10 | 8 | 4 | 3 | 15 | 12 | ÷7 |
| Taille typ.(GB) | 0.001 | 0.01 | 0.160 | 1 | 20 | 160 | 1,500 | 3,000 | 8,000 | ×1,500,000 |

Sources : `https://jcmit.net/` et différents sites web de vendeurs de RAM.

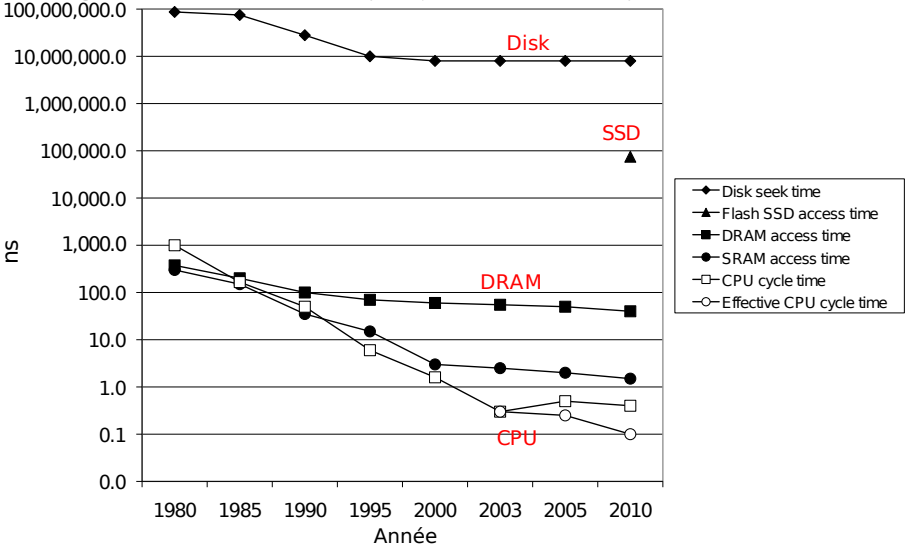
# À propos de la fréquence des CPU INTEL

| | 1980 | 1990 | 1995 | 2003 | 2004 | 2005 | 2010 | 2015 | 2020 | 2020 vs 1980 |
|---|---|---|---|---|---|---|---|---|---|---|
| CPU | 8080 | 386 | Pentium | P-III | P-4 | Core 2 | Core i7 | Core i9 | Core i9 | — |
| MHz | 1 | 20 | 150 | 600 | 3300 | 2000 | 2500 | 3000 | 2900 | ×2500 |
| $T_{cycle}$ (ns) | 1000 | 50 | 6 | 1.6 | 0.3 | 0.50 | 0.4 | 0.33 | 0.35 | ÷2500 |
| Cœurs | 1 | 1 | 1 | 1 | 1 | 2 | 4 | 8 | 16 | ×16 |
| $T_{eff}$ (ns) | 1000 | 50 | 6 | 1.6 | 0.3 | 0.25 | 0.1 | 0.04 | 0.022 | ×45,000 |

Tendance similaire chez les autres constructeurs

# Tendance

## Écart grandissant entre temps cycle CPU et temps accès mémoire



# Plan

# Principe de la hiérarchie mémoire

## Constat

- Larges espaces de stockage lents et lointains
- Petits espaces de stockage proches et rapides

## Idée

- Copier les zones « utiles » de la grosse mémoire dans la petite
- Et le faire automatiquement sans intervention du programme

## Problème

- Collisions !
- Comment faire « tenir » des données d'adresses très variées dans le cache ?

# Pourquoi cela marche-t-il ?

Repose sur une propriété fondamentale des programmes : la *localité*

## Principe de localité

Les programmes tendent à utiliser des instructions et des données dont les adresses sont identiques ou proches de celles qu'ils ont utilisé récemment

## Localité temporelle

Des variables accédées récemment ont de fortes chances d'être accédées de nouveau dans le futur proche

## Localité spatiale

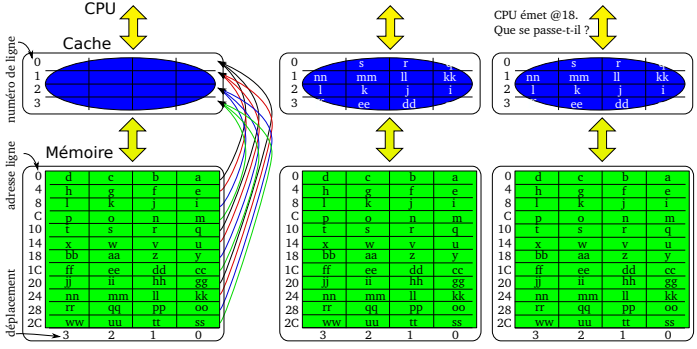
Des variables qui ont des adresses proches ont de fortes chances d'être accédés à des temps proches

# Utilisation de la localité

## Spatiale et temporelle

Lecture d'un mot

1. recopie le mot et ses $n$ voisins (une *ligne*) de la mémoire centrale dans le cache s'il n'y est pas
2. retourne le mot qui est dans le cache



# Architecture d'un cache à correspondance directe

## Principe

Structure : $l = 2^n$ lignes de $m = 2^p$ mots

- on ignore les octets dans le mot :
  les adresses considérés finissent par $00$
- adresse d'une ligne en mémoire :
  alignée sur la taille de la ligne, *i.e.* multiple de $m \times 4$
- numéro de ligne de cache $index$ dans le cache d'une donnée placée en $am$ :
  $index = am_{n+p+1..p+2}$
- adresse $offset$ de la donnée dans la ligne :
  $offset = am_{p+1..2}$
- comment différencier les différentes lignes ?
  nécessaire de conserver $am_{31..n+p+2}$ pour comparaison

# Architecture d'un cache à correspondance directe