Performances et coûts

Pour une technologie de circuits et une architecture données:

- Réduction du nombre de cycles d'horloge nécessaires à l'exécution d'une instruction
- Simplification de l'organisation pour raccourcir le cycle d'horloge
- Chevauchement des instructions exécutées (techniques d'*overlapping*

Les Mémoires

- Mémoire
  - Dispositif capable d'enregistrer, de conserver et de restituer des informations
  - Informations binaires pour un ordinateur
- On classe les mémoires selon
  - Caractéristiques : capacité, débit ...
  - Type d'accès : séquentiel, direct ...

Caractéristiques des mémoires

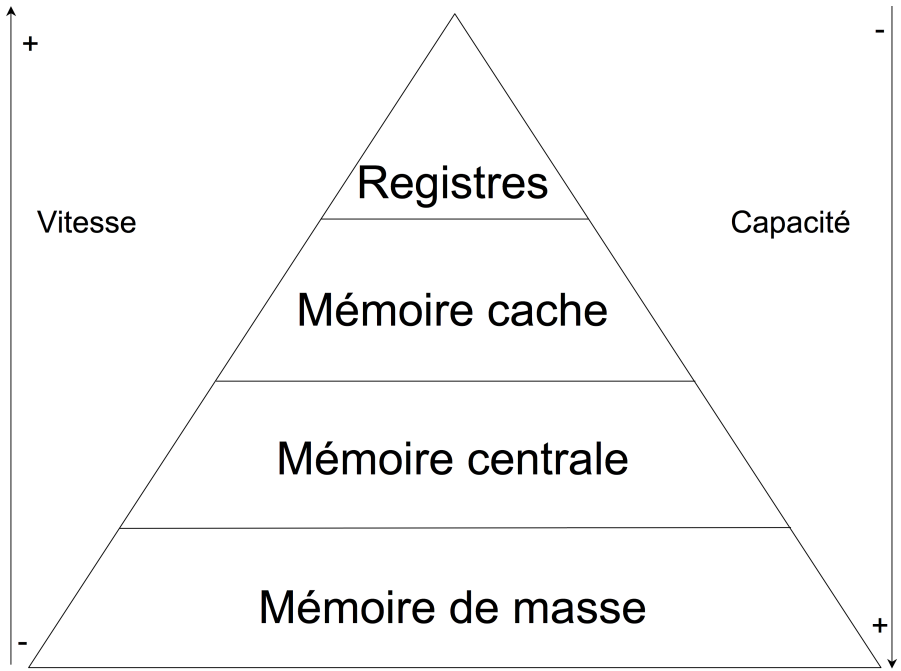
- Temps d'accès
  - Temps s'écoulant entre le lancement d'une opération de lecture/écriture et son accomplissement
- Cycle mémoire
  - Temps minimal entre 2 accès successifs à la mémoire
  - Cycle > temps d'accès
  - Car besoin d'opérations supplémentaires entre 2 accès (stabilisation des signaux, synchronisation ...)
- Débit
  - Nombre d'informations lues ou écrites par seconde
  - Exemple : 300 Mo/s
- Volatilité
  - Conservation ou disparition de l'information dans la mémoire hors alimentation électrique de la mémoire

Méthodes d'accès

- Accès séquentiel
  - Pour accéder à une information on doit parcourir toutes les informations précédentes
  - Accès lent
  - Exemple : bandes magnétiques (K7 vidéo)
- Accès direct
  - Chaque information a une adresse propre
  - On peut accéder directement à chaque adresse
  - Exemple : mémoire centrale
- Accès semi-séquentiel
  - Intermédiaire entre séquentiel et direct
  - Exemple : disque dur
    - Accès direct au cylindre
    - Accès séquentiel au secteur sur un cylindre
- Accès associatif/par le contenu
  - Une information est identifiée par une clé
  - On accède à une information via sa clé
  - Exemple : mémoire cache

Hiérarchie Mémoire

Dans un ordinateur, plusieurs niveaux de mémoire:



Registres

- Se trouvent intégrés dans le CPU
- Un registre est un mot stockant des informations relatives à une instruction :
  - Opérandes
  - Paramètres
  - Résultats
- Peu nombreux dans un CPU
- Très rapides (vitesse du CPU)
- Voir le cours sur les processeurs

Mémoire cache

- Mémoire intermédiaire entre le processeur et la mémoire centrale:
- Lq Mémoire cache est intégrée dans le processeur et est cadencée à la même fréquence
- But de la mémoire cache :
  - Débit de la mémoire centrale très lent par rapport au débit requis par le processeur
  - On accélère la vitesse de lecture des informations par le CPU en les plaçant (en avance) dans le cache
- Mémoire associative
- De type SRAM car doit être rapide
- Taille : de quelques centaines de Ko à quelques Mo

Mémoire centrale

- Taille : quelques centaines de Mo à quelques Go
- Accès direct
- De type DRAM car moins cher
- Vitesse relativement lente

Comparaison vitesse cache/centrale

- Mémoire : SDRAM-DDR 2100
- Processeur : AMD Athlon XP 2200+ (1.8 Ghz)
- Dans les 2 cas : lecture de mots de 64 bits
- Mémoire
  - Fréquence de 133 Mhz et mots de 64 bits
  - 2 accès par cycle horloge (DDR = Double Data Rate)
  - Débit théorique maximum de 2,1 Go/s (moins en pratique)
- Processeur
  - Cache L1 du processeur : débit mesuré de 18 Go/s
  - Cache L2 du processeur : débit mesuré de 5,6 Go/s

Mémoire de masse

- Mémoire de grande capacité : plusieurs centaines de Mo à plusieurs centaines de Go
- Mémoire non volatile (Stockage)
- Très lente
- Exemples
  - Disque dur
  - Bande magnétiques
  - DVD ou CD

Hiérarchie mémoire : conclusion

- Organisation de façon à ce que la CPU accède le plus rapidement possible aux données les plus utilisées
- Hiérarchie
  - Mémoire cache : rapide et petit
  - Mémoire centrale : moins rapide et plus gros
  - Mémoire de masse : lent et très gros
- Plus une mémoire est lente, moins elle est chère

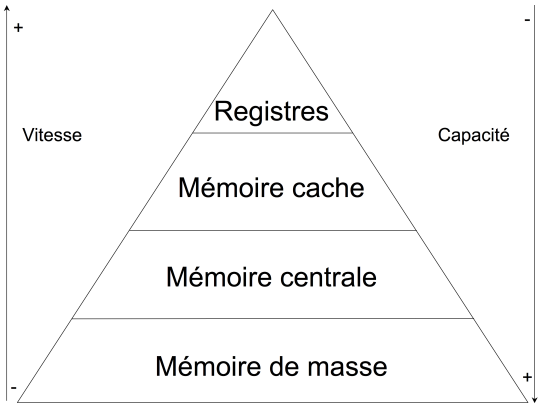
Mémoire Cache

Problème: La mémoire est beaucoup plus lente que le processeur (facteur 10):

- plusieurs instructions par cycle d'horloge (ie. chaque nanoseconde)
- quelques dizaine de nanosecondes pour accéder à la mémoire.

Les mémoires aussi rapides que le processeur coûtent trop cher.

Solution: Une mémoire à plusieurs niveaux



Mémoire Cache

- Problèmes
  - Mémoire cache doit être petite (quelques centaines de Ko ou quelques Mo) pour être efficace en terme de débit
  - Ne peut donc pas y stocker tout un programme et ses données
- Solutions
  - Algorithmes pour « deviner » et mettre dans le cache les données/instructions avant que le CPU en ait besoin
  - Recherche bon compromis entre tailles, types de cache (données/instructions), niveaux de cache, techniques d'accès au cache ... pour meilleures performances

Localité et pre-fetching

- Pour améliorer le taux de succès du cache
  Pre-fetching : chargement en avance des données dont le CPU devrait avoir besoin
- Algorithmes de pre-fetching sont basés sur les principes de localité:
  - Localité temporelle
    Garder en mémoire cache les dernières données référencées par le programme
  - Localité spatiale
    Charger en avance les données/instructions contigues à une donnée/opération référencée

Localité

Définition

Les programmes standard ont souvent des comportements prédictibles soit temporellement, soit spatialement, de sorte que les accès mémoires réalisés par le processeur pour les exécuter ne se font pas au hasard.

Localité temporelle

- Une donnée référencée à un temps *t* aura de très fortes chances d'être référencée dans un futur proche:

Principe de localité temporelle

Le processeur fait plusieurs fois référence à une même case mémoire à des instants rapprochés.

- Il est interressant de stocker dans le cache une donnée à laquelle le processeur à récemment accédée.
- Ce principe n'est pas vrai à 100% !

Localité spatiale

- Si une donnée est référencée à un temps t, alors il y a de très fortes chances que les données voisines le soient dans un futur proche
- Exemple :

```
for (i=0; i < N; i++)
    somme += A[i];
```

Principe de localité spatialle

Si le processeur accède à une case mémoire à un instant donné, il accèdera probablement à des cases mémoires voisines aux instants suivants.

- On ramène dans le cache, l'information qui ne s'y trouve pas + un ensemble (appelé ligne) de plusieurs cases mémoires voisines.
- Ce principe n'est pas vrai à 100% !

Dimension Globale

- Augmentation de la taille de la mémoire cache
- Augmente le taux de succès
- Mais ne gagne pas forcément en performance car augmentation du temps d'accès proportionnellement à la taille

Accès par lignes mémoire

- Mémoire cache = sous-partie de la mémoire centrale
- Comment est déterminée cette sous-partie ?
- Principe
  - Les échanges d'informations entre mémoire de niveaux différents se font par blocs d'adresses consécutives: une ligne mémoire
  - pas beaucoup plus coûteux de lire une ligne qu'un seul mot
  - et s'adapte bien au principe de localité spatiale

Accès par lignes mémoire

- Pour le processeur la présence de la mémoire cache est transparente
- Le CPU demande toujours à accéder à une adresse en mémoire centrale (pour lecture ou écriture)
- Adressage d'un mot mémoire
  - Adressage direct : adresse du mot en mémoire
  - Adressage via ligne : index de la ligne et déplacement dans la ligne, index de n bits et un déplacement de m bits.
  - Correspondance entre les 2 adresses
    $adresse\ memoire = index \times taille\_ligne + dplacement$

Adressage

- La mémoire cache contient des lignes de mots de la mémoire centrale
- Trois méthodes pour gérer la correspondance entre une ligne dans le cache et une ligne de la mémoire centrale
  - Correspondance directe
  - Correspondance associative totale
  - Correspondance associative par ensemble
- Exemples:
  - Lignes de 32 octets
  - Mémoire cache de 512 Ko : 16384 lignes
  - Mémoire centrale de 128 Mo : doit être gérée via les 512 Ko de cache et ses 16384 lignes

Cache direct

- L lignes en cache
- La ligne d'adresse j en mémoire centrale sera gérée par la ligne i en cache avec $i = j\%L$
- A partir de l'adresse d'une ligne en mémoire on sait directement dans quelle ligne du cache elle doit se trouver
- Exemple :
  - Chaque ligne du cache correspond à :
    128 x 1024 x 1024 / 16384 = 8192 octets = 256 lignes
  - Une ligne i du cache contiendra à un instant donné une des 256 lignes j de la mémoire tel que $i = j\%16384$

Avantages/Inconvénients du cache direct

Avantages

- On sait immédiatement où aller chercher la ligne
- Accès très rapide à la ligne (latence d'1 cycle d'horloge)

Inconvénient

- Devra parfois décharger et charger souvent les mêmes lignes alors que d'autres lignes sont peu accédées
- Peu efficace en pratique
- Taux de succès entre 60 et 80 %

Cache complètement associatif

Définition

Une ligne de la mémoire cache correspond à n'importe quelle ligne de la mémoire centrale

Avantages/Inconvénients du cache associatif

Avantages

- Très souple et efficace pour gérer les lignes de manière optimale en terme de succès d'accès

Inconvénient

- Doit au pire parcourir toutes les lignes du cache pour savoir si la ligne cherchée s'y trouve ou pas

Au final

- Taux de succès de l'ordre de 90 à 95%
- Mais temps d'accès de l'ordre de $(L/2)$ cycles d'horloge

Cache mixte ou associatif par ensemble

Définition

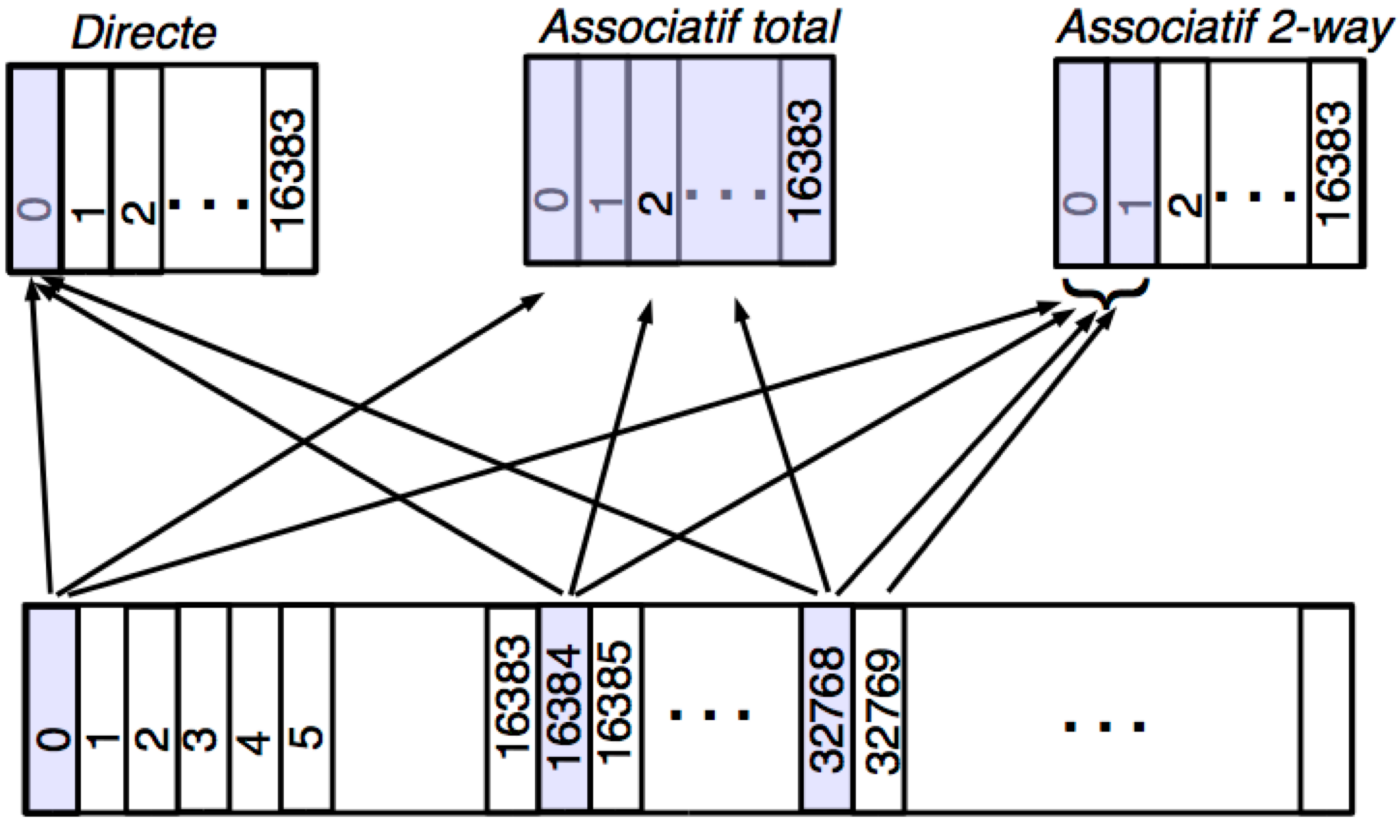
- Correspondance associative par ensemble (N-way associative)
- Solution intermédiaire aux 2 précédentes
- On regroupe les lignes du cache en ensembles de N lignes
- Une ligne de la mémoire centrale est gérée par un ensemble donné donc peut être une de ses N lignes

Avantages/Inconvénients du cache mixte

Avantages

- Plus souple et efficace que la correspondance directe
- Temps d'accès court = 2.5 cycles pour N = 4
- Taux de succès : 80% à 90%
- Méthode utilisée en pratique car meilleur compromis

Correspondance lignes cache/mémoire



Remplacement des lignes

Le cache contient une partie des lignes de la mémoire centrale

- Si lecture ou écriture par le CPU dans une ligne qui n'est pas dans le cache
- On doit charger en cache cette ligne
- Nécessité d'enlever des lignes pour y mettre d'autres lignes
- 5 méthodes pour choisir la ligne à remplacer :
  - Aléatoire
  - First In First Out
  - Least Recently used
  - Least Frequently used
  - NMRU

Remplacement Aléatoire

- Simple à mettre en oeuvre
- Peu efficace en pratique car on peut supprimer des lignes très accédées.

Remplacement FIFO

First In First Out

- Premier entré Premier sorti
- Idée: les informations trop vieilles dans le cache ne vont plus servir
- chaque ligne du cache contient sa date d'entrée, la plus vieille est remplacée par une ligne de la mémoire principale
- statistiquement : plus efficace que le précédent
- défaut : une ligne référencée régulièrement est retirée du cache quand elle devient la plus vieille

Remplacement LRU

Last Recently Used

- Idée: l'important n'est pas l'ancienneté mais l'ancienneté d'utilisation
- on associe à chaque ligne la date de dernière utilisation
- on extrait la ligne la plus vieille
- statistiquement : le plus efficace
- Inconcevénient : complexité accrue (modification de la date à chaque accès)

Remplacement LFU

Least Frequently Used

- Variante de LRU
- on associe à chaque ligne un compteur s'incrémentant à chaque utilisation
- on extrait la ligne dont le compteur est le plus petit
- Tri plus simple sur les compteurs que sur les dates

Remplacement des lignes

- Cache direct: pas besoin de choisir on ne peut remplacer qu'une seule ligne
- Cache associatif: toutes les méthodes de remplacement peuvent être utilisées
- Cache mixte:
  - On utilise une approche intermédiaire simple entre aléatoir et LRU
  - dans chaque ensemble de N lignes du cache on marque la dernière ligne accédée
  - on remplace aléatoirement l'une des $N - 1$ lignes non marquées.

Remplacement NMRU

- Variante de LRU
- On s'assure que la ligne la plus récente reste dans le cache
- chaque ligne a un bit supplémentaie qui est mis à 1 lorsque la ligne est utilisée et à 0 sinon
- on remplace aléatoirement parmi les lignes à 0
- résultats satisfaisant ...

Politique de Réécriture

Outre les opérations de lecture en mémoire principale, le processeur effectue également des écritures, par exemple lorsqu'il modifie des données.

Cache write-through

- Écriture simultanée (write-through)
- Quand on écrit un mot d'une ligne du cache, on écrit simultanément le mot de la même ligne en mémoire centrale
- Écriture plus longue car accès mémoire centrale

Cache write-back

- Écriture différée ou réécriture (write-back)
- On n'écrit le mot que dans la ligne du cache
- Quand cette ligne est supprimée du cache, on l'écrit en mémoire centrale

Intérêt/Inconvénients du write-back

Avantages

- Limitation des écritures en mémoire centrale

Inconvénients

- Problème si d'autres éléments accèdent à la mémoire en écriture
- Périphériques en mode DMA (Direct Memory Access)
- Multi-processeurs avec mémoire commune
- Nécessite alors des algorithmes supplémentaires pour gérer la cohérence

Temps de récupération d'une instruction

- Temps d'accés au cache T_c
- Temps de pénalité au cache T_m
- Taux d'échec du cache T_{echec}
- Temps moyen d'accés pour une instruction :

$$T_{acces} = T_{echec} T_m + (1 - T_{echec}) T_c$$

- L'amélioration de l'un de ces termes améliore le temps moyen d'exécution

Causes des défauts de cache

Différents types

- Échec obligatoire
- Échec de capacité
- Échec de conflit

Hiérarchiser la mémoire: caches multi-niveaux

Organisation en niveau

- Cache est organisé en plusieurs niveaux
- Niveaux L1 et L2, voire L3 pour certains processeurs
- Cache Li+1 joue le rôle de cache pour le niveau Li
- Cache Li+1 plus grand que Li mais moins rapide en temps d'accés aux donnés
- Cache L1 : généralement scindé en 2 parties Instructions/Données

Organisation en niveaux

Relation entre les niveaux de cache

- Cache inclusif
  - Le contenu du niveau L1 se trouve aussi dans L2
  - Taille globale du cache : celle du cache L2
- Cache exclusif
  - Le contenu des niveaux L1 et L2 sont différents
  - Taille globale du cache : taille L1 + taille L2

Avantages et inconvénients de chaque approche

Exclusif

- Cache plus grand au total
- L2 de taille quelconque
- Mais doit gérer la non duplication des données : prend du temps, L2 moins performant

Inclusif

- Cache L2 plus performant
- Mais taille totale plus faible
- Et contraintes sur la taille de L2 (ne doit pas être trop petit par rapport à L1)
- Cache inclusif : historiquement le plus utilisé
- Mais trouve aujourd'hui des CPU à cache exclusif (AMD)

Performance

- Evaluation de la performance d'une mémoire cache: Taux de succès le plus élevé possible
  - Succès : la donnée voulue par le CPU est dans le cache L1
  - Ou la donnée voulue par le cache L1 est dans le cache L2 ...
  - Echec : la donnée voulue n'y est pas
  - Doit la charger à partir du niveau de cache suivant ou de la mémoire centrale
- A prendre en compte également: Temps d'accès à la mémoire, latences : nombre de cycle d'horloges requis pour
  - Lire une donnée dans le cache en cas de succès
  - Aller la récupérer en cas d'échec au niveau supérieur
  - Latences dépendent de la taille des niveaux et de l'organisation des données dans les niveaux

Performance : niveaux

- Choix de taille du cache et du nombre de niveaux
- Augmentation du nombre de niveaux (plus de 3)
- Pas de gain supplémentaire vraiment significatif
- Cache inclusif
- Le cache L2 doit être bien plus grand que le cache de niveau L1 car sinon on stocke peu de données supplémentaires dans L2 par rapport à L1 et donc taux de succès de L2 faible
- Ne pas oublier le coût élevé de ce type de RAM

Conclusion

Les performances globales de la mémoire cache dépendent de plusieurs facteurs corrélés

- Taille de la mémoire cache et organisation des niveaux
- Méthode d'association des lignes entre mémoire centrale et cache
- Rapidité d'accès à la bonne ligne dans le cache
- Méthode de remplacement des lignes
- Algorithmes de pre-fetching
- Chargement en avance dans le cache de données/instructions dont le processeur devrait avoir besoin

But : un taux de succès global qui doit être le plus élevé possible tout en assurant un temps d'accès bas (Généralement autour de 90% de nos jours)