Hiérarchie mémoire : structure et applications

problème: temps de fonctionnement
répertoires ≤ 1 cycle | RAM ≈ 500 cycles
comment rendre la mémoire plus réactive ?

idée: dupliquer partiellement la mémoire dans un cache plus petit, proche et réactif. Ainsi le contenu qui y est stocké peut être accédé et modifié beaucoup plus vite.
(localité temporelle: on manipule régulièrement les mêmes données)

I structure du cache

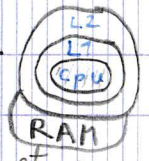
1) structure en bloc.

- Localité spatiale : quand on accède à une adresse on accède aux adresses contigus.

↳ le cache charge et évince le contenu de la mémoire par blocs

2) structure en oignon

- Implanter plusieurs cache de taille croissante et de réactivité décroissantes traversé les uns après les autres lors de l'accès et de l'écriture dans la mémoire.



3) transparence.

- Le processeur accède et écrit dans la mémoire et ne sait pas si ses requêtes sont interceptés par un cache.
- Nécessité : Le fonctionnement du cache est caractérisé par le matériel.

4) Métadonnées

5) Performance

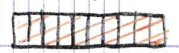
AMAT = Temps de recherche dans le cache + Taux d'échec x (temps moyen en cas d'échec)

But : AMAT << temps moyen d'accès sans cache.

II Choix d'implémentation :

1) associativité

- fully associative : n'importe quel bloc peut être mis n'importe où dans le cache. Très lourd à implémenter physiquement.



- directly mapped : un bloc ne peut être placé qu'à une seule position du cache dépendant de son adresse. Les adresses ne sont pas bien réparties et donc le même bloc du cache est utilisé en boucle.

- Set associative : pour chaque bloc de la mémoire il y a un ensemble de positions dans le cache où il peut être placé

Intermédiaire

2) types d'adresses

On peut associer les blocs à leurs adresses physiques ou bien virtuelles. Dans le 1er cas, tester la présence du bloc dans le cache nécessite un appel à la MMU. Dans le second, le cache doit être purgé à chaque commutation.

3) Politique d'écriture et d'allocation

a) écriture

- Write-through : chaque écriture est propagée en mémoire.
- Write-back : les écritures sont locales au cache. Elles ne sont propagées que lors de l'éviction de la ligne

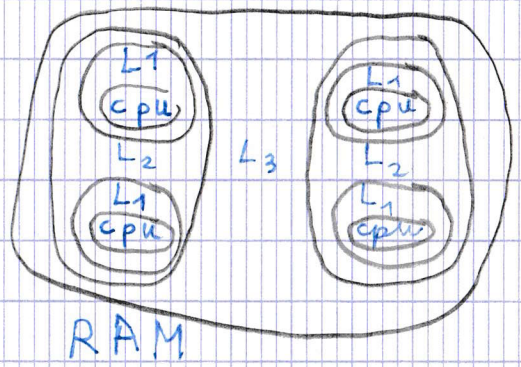
b) allocation:

- Write allocate: Lors d'un echec d'ecriture, la ligne est chargée.
- Write non-allocate: Lors d'un echec d'ecriture, la ligne n'est pas chargée.

4) multi'coeur.

a) localité du cache.

- un cache par coeur: Plus cher, des caches plus petits.
- un cache par puce: Plus loin, moins reactif.



b) Coherence.

- Les données dans un cache peuvent différer des données dans un autre cache ou dans la mémoire (charger une donnée qu'un autre coeur modifie dans la mémoire / ecrire dans son cache mais pas dans la mémoire).
- Def 1: Protocole MESI.

III Problematiques liées

1) Optimisations

a) separer le cache donnée / cache instruction

- Les données et les instructions fonctionnent très differement lors de l'execution d'un programme. Ainsi on separe le cache en 2, un cache de donnée et un cache d'instruction utilisant des stratégies de caches très differentes.

b) préchargement

- On essaye de deviner le prochain bloc utilisé et on le charge à l'avance. Favoriser la localité spatiale au prix de la localité temporelle: prechargement séquentiel. Favoriser la régularité d'une execution: prechargement à pas.

c) Victim cache.

- Coller au cache L_1 un cache associatif, le victim cache qui ramene les blocs evincé du cache L_1. Presque aussi rapide que le cache L_1. Il ne ralentit pas les echecs du cache L_1. Diminue le problème lié au mauvaise repartition des adresses.

d) optimiser le code pour le cache.

- Lors de la compilation, on optimise le code obtenu pour qu'il tire au mieux avantage du cache: Ex: permutation de boucle.

2) Sécurité Dev 2: attaque Spectre.

3) Autres formes de cache:

a) TLB

- ici on veut gagner du temps en évitant un calcul par la MMU.

b) SSD cache

- utiliser un SSD (plus rapide) comme cache pour un banc de disques durs

c) SQL buffer cache

- stocker le résultat de requêtes SQL clés pour éviter leur calcul et simplifier les transactions.