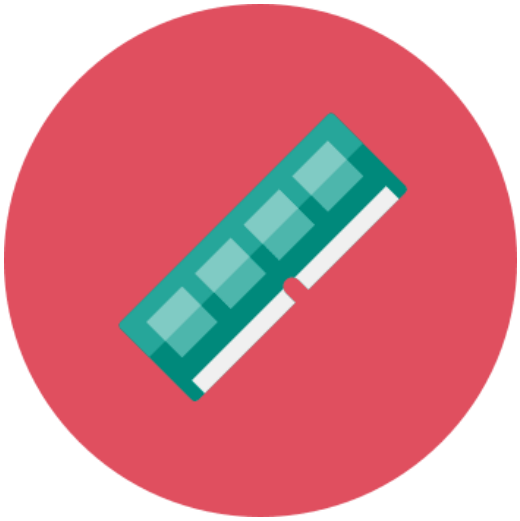
Les mémoires cache



Christopher MILAZZO

Licence STS informatique 2019 – 2020

Table des matières

1 Généralisation

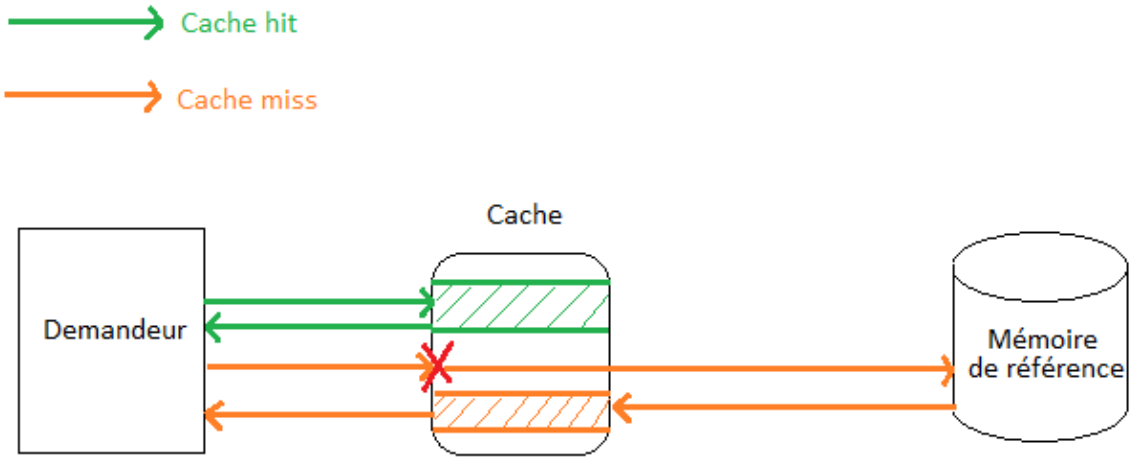
De manière générale, la mémoire cache (ou l'antémémoire en Français), permet d'économiser et de réduire le temps d'accès aux ressources. Par exemple le CPU qui souhaite accéder aux données contenues en mémoire centrale.

2 Principe de fonctionnement

L'antémémoire se positionne entre un élément « demandeur » et une « mémoire de références » contenant les ressources (les données). Elle est de très petite capacité et très onéreuse. Par définition l'accès à l'antémémoire est beaucoup plus rapide.

Lorsque le demandeur a besoin d'accéder à une ressource en écriture ou en lecture, le principe de fonctionnement est le suivant :

1. L'antémémoire vérifie qu'elle ne contient pas une copie de la ressource réclamée. Si la ressource est contenue, l'antémémoire la fournie directement au demandeur. On parle alors de ***succès de cache*** (Cache hit en anglais) (voir flèche verte sur le schéma plus bas). En revanche si la ressource n'y est pas, on parle alors de ***défaut de cache*** (Cache miss) (voir flèche orange sur le schéma plus bas). Le cas échéant, on procède aux étapes suivantes…
2. La demande se propage au niveau de mémoire supérieur. Cette dernière renvoie la ressource souhaitée.
3. L'antémémoire stocke la ressource demandée pour une utilisation ultérieure puis la retransmet à l'élément demandeur.



Ce qui détermine la performance d'une antémémoire est ce que l'on appelle le taux de succès (Hit ratio) qui est le nombre de succès de cache par le nombre d'accès mémoire.
Il est évident que plus celui-ci est élevé, plus efficace sera l'antémémoire, et plus on aura économisé du temps d'accès aux ressources.

Les défauts de cache se distinguent en 4 types :

- les défauts de cache obligatoires : ils correspondent à la première demande du processeur pour une donnée/instruction spécifique et ne peuvent être évités ;
- les défauts de cache capacitifs : l'ensemble des données nécessaires au programme excèdent la taille du cache, qui ne peut donc pas contenir toutes les données nécessaires ;
- les défauts de cache conflictuels : deux adresses distinctes de la mémoire de niveau supérieur sont enregistrées au même endroit dans le cache et s'évincent mutuellement, créant ainsi des défauts de cache (voir les caches directement adressés plus bas) ;
- les défauts de cohérence : ils sont dus à l'invalidation de lignes de la mémoire cache afin de conserver la cohérence entre les différents caches des processeurs d'un système multi-processeurs.

3 Les types de cache

On distingue deux types de mémoire cache. Le cache matériel essentiellement (ou totalement) implémentée de manière électronique, ou bien le cache logiciel, implémenté directement par un programme informatique.

3.1 Les caches matériels

Une mémoire cache matériel est très performante mais aussi très onéreuse. Un exemple de cache matériel très utilisé est la SRAM (Static Random Access Memory). La SRAM est un type de mémoire vive utilisant des bascules pour mémoriser l'information.

Une SRAM de 1999 :

3.2 Les caches logiciels

L'un des caches logiciel le plus connu est le cache implémenté par les navigateurs WEB. Lorsque qu'une page web est chargée, il est fréquent que celle-ci requière un certain nombre de fichiers annexes (images, javascript, css, etc.). Afin de limiter le nombre de requêtes et d'améliorer la vitesse de chargement des pages ultérieurement, les fichiers annexes sont stockés sur la mémoire de masse de l'ordinateur et seront réutilisés à la demande.

Un autre exemple concerne le SGF d'un système d'exploitation. Lorsqu'un fichier est ouvert pour lecture et/ou écriture, celui-ci est en réalité mis dans un cache logiciel sur la mémoire centrale afin de rendre son accès plus rapide.

4 Problématiques d'implémentation

La conception des caches implique de prendre en compte plusieurs problématiques listées ci-après :

4.1 Notion d'optimisation

La raison d'être de l'antémémoire est de permettre l'accès aux ressources plus rapidement afin d'améliorer la vitesse d'exécution des programmes.

Deux grands principes ont pu être mis en lumière lors d'étude comportementale des programmes informatiques :

- La localité spatiale, qui indique que l'accès à une ressource située à l'adresse X, sera probablement suivi d'un accès à la ressource située à une adresse X+1 ;
- La localité temporelle, qui indique que l'accès à une certaine zone mémoire à un instant T, a de forte chance de se reproduire dans un lapse de temps très cours (principe des traitements itératifs).

Ainsi, la conception des mémoires cache va tendre à tirer un maximum profit de ces principes. Certaines implémentations cherchent même à améliorer le taux de succès (hit ratio) en prédisant à l'avance (1 ou 2 cycles CPU) la donnée à charger en cache. Pour des programmes correctement optimisés, la plupart des prédictions seront correctes, ce qui rend l'implémentation pertinente.

Enfin il existe également une technique qui consiste à bypasser le cache pour les données qui sont rarement demandées.

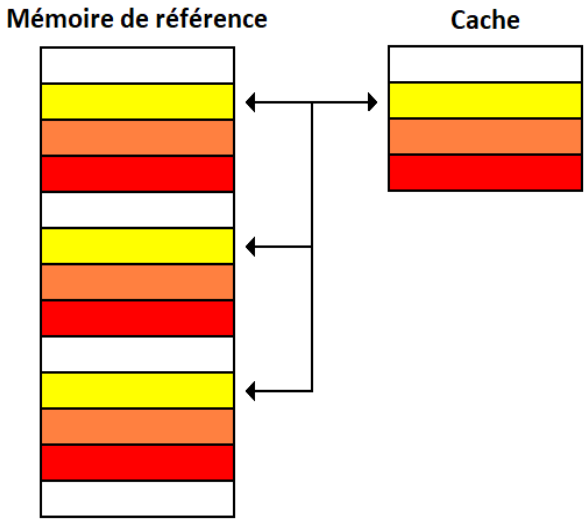
4.2 Placement de l'information

Le choix de la façon d'associer le contenu du cache avec le fournisseur est primordiale, car les performances en dépendent.

On distingue 4 types de caches :

- Directement adressés (direct mapped)
- Totalement associatifs (fully associative)
- Associatifs par voie (N-Way associative)
- Pseudo-associatifs

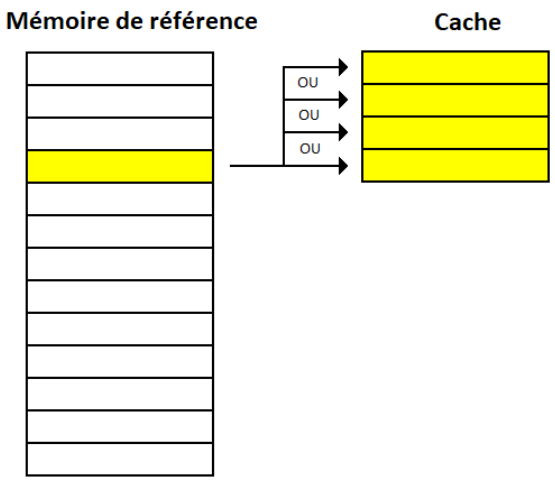
4.2.1 Cache directement adressé



Les adresses de la mémoire de référence sont congrues modulo n (où n est la capacité du cache) en suivant un mapping prédéfini.

Avantages
Le parcours de la mémoire est rapide puisque l'on connait d'avance où est susceptible d'être stockée la ressource en cache. Donc un temps d'accès à la ressource court.
Inconvénients
Du fait que les cases mémoires ont un emplacement en cache prédéfini, le défaut de cache conflictuel a plus de probabilité d'arriver.
Le taux de succès est faible, et n'est pas suffisamment comblé par le temps d'accès court.

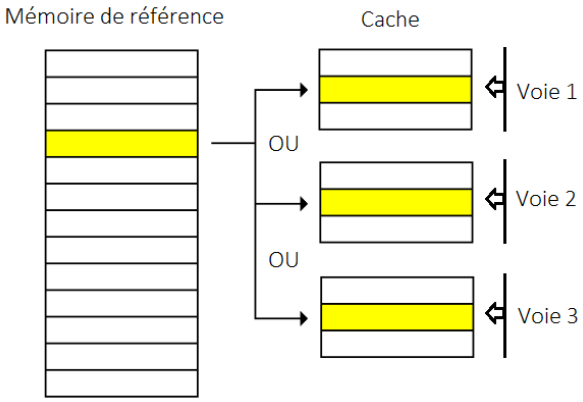
4.2.2 Cache totalement associatif



Dans cette implémentation, les adresses mémoires sont mappées n'importe où sans restriction dans la mémoire cache.

Avantages
Pas de défaut de conflit, donc taux de succès élevé.
Inconvénients
Temps d'accès à la ressource élevé, car il faut parfois parcourir toute la mémoire cache avant de tomber éventuellement sur la ressource demandée.

4.2.3 Cache associatif par voie



Les deux implémentations précédentes ont chacune leurs avantages. Ces avantages sont complémentaires, l'idéal serait de trouver un juste milieu afin de bénéficier des deux avantages tout en diminuant les inconvénients.

C'est justement ce vers quoi tend le cache associatif par voie. Le principe est de diviser la mémoire cache en plusieurs petites mémoires appelées « voie » et de faire un mapping comme le cache totalement associatif pour chacune d'elles.

On maintient un bon taux de succès en limitant les défauts conflictuels en alternant l'allocation entre les voies, et on réduit le temps de d'accès car on connait l'emplacement potentiel de la ressource, la seule variable est la voie dans laquelle est éventuellement stockée la ressource.

Certaines mémoires cache associatives tendent à accélérer le temps d'accès en faisant des prédictions de voie.

4.2.4 Cache pseudo-associatif

Fonctionne comme un cache associatif par voie, mais accède aux voies une par une en séquence et non en parallèle comme le fait le cache associatif par voie. L'opération est plus rapide dans le meilleur des cas, sinon il est plus long car il faudra vérifier toutes les voies une par une.

4.3 Politique de remplacement

Par définition une antémémoire est de petite taille. Par conséquent, arrive une situation où lors d'un cache miss il n'y a plus de place pour accueillir la nouvelle ressource. L'antémémoire doit se débarrasser d'une des ressources préalablement mises en cache afin de la remplacer.

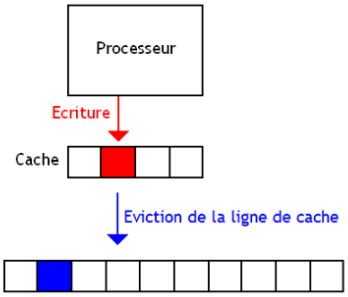
En fonction du contexte d'utilisation du cache, le choix va se porter sur un algorithme ou un autre. Il en existe un certain nombre (FIFO, LRU, etc…), mais ce document ne traite pas ce sujet.

Il est à noter qu'on retrouve ces algorithmes dans d'autres utilisations que pour l'antémémoire. Par exemple, ils sont utilisés par le système d'exploitation pour gérer la pagination de la mémoire centrale.

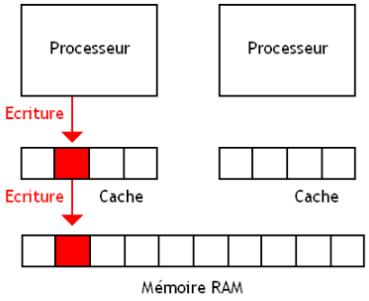
4.4 Gestion des mises à jour des données

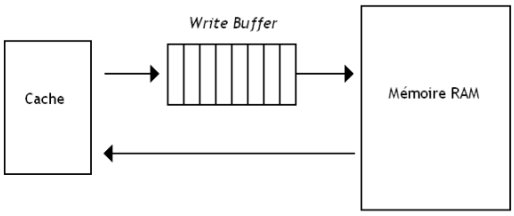
Suivant les besoins et le niveau du cache il existe deux méthodes d'écriture :

<u>Le write-back :</u>
Lors de l'écriture de données, celles-ci seront copiées en mémoire uniquement lorsque la ressource concernée sera chassée du cache suite à un conflit de capacité. Cette technique permet de limiter l'écriture et la lecture des données en mémoire centrale.



<u>Le write-through :</u>
Lors de l'écriture de données, la copie de celle-ci se fera immédiatement au travers de toutes les couches de la mémoire cache jusqu'à la mémoire de référence. Cette technique a l'inconvénient de causer beaucoup d'écriture en mémoire, mais en revanche elle réduit la complexité de gestion des incohérences de cache en les maintenant à jour en permanence.





Afin d'éviter d'avoir un cache bloquant lors d'éviction de ligne de cache, il peut être mis en œuvre un buffer d'écriture. Celui-ci permet de ne pas devoir attendre la fin de l'écriture de la donnée en mémoire centrale avant de libérer la place en mémoire cache. Le buffer fonctionne comme une liste en FIFO.

4.5 Dimensionnement

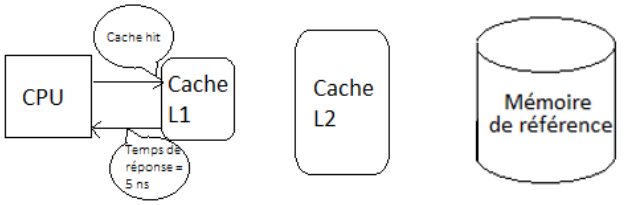
De manière très théorique et à relativement petite échelle, on peut dire que plus on augmente la taille d'une mémoire cache, plus l'accès aux ressources sera performant.

Toutefois cela devient faux à partir d'un certain point. Il existe une taille à partir de laquelle l'augmentation de la capacité de la mémoire cache devient inutile. Cela réside dans le fait que la structure d'un programme est constituée de plusieurs embranchements qui ne sont pas prédictibles par le CPU. Chaque embranchement pointe sur une zone mémoire différente. Ainsi, l'optimisation par anticipation d'accès aux ressources (voir 4.1) devient obsolète.

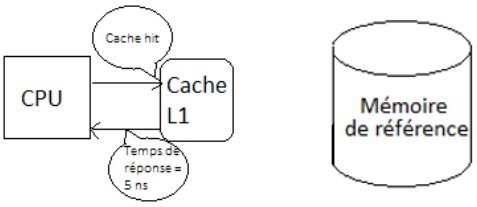
4.6 Hiérarchie des caches

Il est courant d'avoir plusieurs niveaux de caches.

En effet, la principale raison d'utiliser plusieurs niveaux réside dans la volonté d'augmenter la valeur moyenne du taux de succès des caches. Lorsque le la ressource n'est pas présente en cache, il faut la réclamer à la mémoire centrale qui prend plus de temps à répondre. Si l'on place un second niveau, on se donne alors comme une seconde chance de pouvoir accéder à la ressource directement depuis l'antémémoire. Voir schéma comparatif ci-dessous :

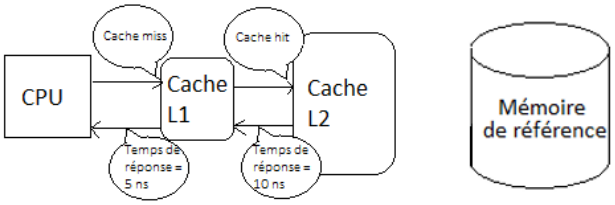


Pas de différence

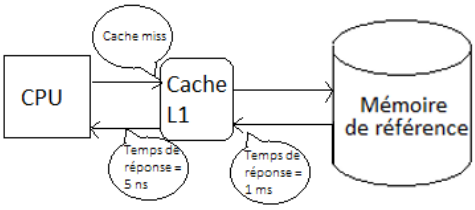


Temps de réponse total = 5 ns

Temps de réponse total = 5 ns

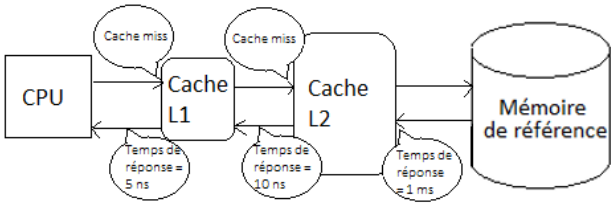


Différence = 900 995 ns

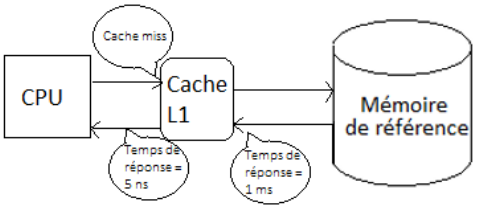


Temps de réponse total = 15 ns

Temps de réponse total = 1 000 005 ns



Différence = 10 ns

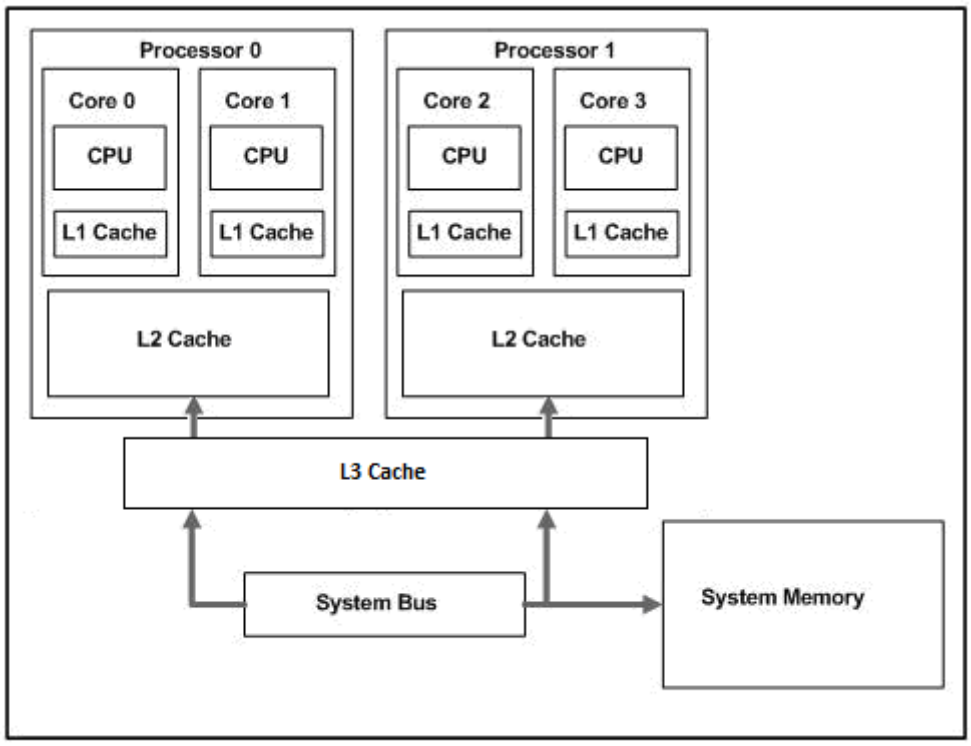


Temps de réponse total = 1 000 015 ns

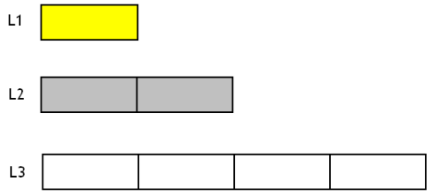
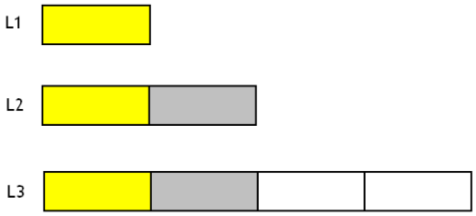
Temps de réponse total = 1 000 005 ns

Autre avantage à avoir plusieurs niveaux cache, notamment pour les architectures multiprocesseurs, est de cloisonner les antémémoires de sorte à limiter les accès concurrents aux caches et d'éviter la génération de défaut de cohérence des mémoires caches. De plus, la hiérarchie des caches permet également de créer des espaces de stockage partagés si besoin est.

Ci-dessous un exemple de système équipé de deux processeurs chacun muni de deux cœurs :



Les cœurs ont chacun accès à un cache L1 qui lui est propre. Au sein d'un processeur, les deux cœurs peuvent partager des ressources au niveau du cache L2. Pour finir le cache L3 permet de stocker des ressources communes aux deux processeurs.

Les caches exclusifs :	Les caches inclusifs :
Chaque niveau contient des ressources différentes. La capacité du cache est utilisée à 100%. Le taux de succès de cache est élevé. Toutefois, le temps d'accès est quelque peut réduit puisqu'il faut parfois parcourir tous les caches pour trouver la ressource recherchée. 	Chaque niveau de cache contient les données du cache supérieur. Avec ce genre de cache, le taux de succès est moins élevé et la capacité globale du cache se trouve réduit. Mais en contrepartie ce genre de comportement permet le partage des ressources plus facilement entre plusieurs demandeurs (plusieurs CPU par exemple). 

5 Références

Fonctionnement mémoires cache :

https://fr.wikipedia.org/wiki/M%C3%A9moire_cache

https://fr.wikibooks.org/wiki/Fonctionnement_d%27un_ordinateur/Les_m%C3%A9moires_cache

La SRAM :

https://fr.wikipedia.org/wiki/Static_Random_Access_Memory