Module 6: Mémoire cache

1 Introduction

Le progrès technologique des ordinateurs a fait en sorte que la mémoire principale n'arrive plus à suivre le rythme croissant des fréquences d'horloge des processeurs. Donc, pour faire face à cette lenteur, il a été possible d'intercaler entre la mémoire principale et le processeur une mémoire cache plus rapide (jusqu'à 10 fois plus rapide que la RAM) et de faible taille (jusqu'à 100 fois plus petite que la RAM). Cela constitue un premier niveau de hiérarchisation de la mémoire.

L'objectif des mémoires caches est de permettre au processeur de mémoriser les informations les plus utilisées afin d'être en mesure de les récupérer le plus rapidement possible en cas de besoin. De cette façon, le processeur n'a pas besoin d'aller chercher les informations auprès de la mémoire principale, ce qui constitue un gain de temps très important.

Bien évidemment, l'efficacité d'un tel système dépend des caractéristiques de la mémoire cache, à savoir son efficacité et sa taille. Car, il est possible de hiérarchiser la mémoire en plaçant plusieurs autres niveaux de cache entre la mémoire principale et le processeur. Dans les systèmes actuels, nous n'avons pas une seule mémoire, mais plutôt plusieurs mémoires organisées en hiérarchie.

2 Principe de la mémoire cache

Vous savez maintenant l'objectif des mémoires caches, elles doivent être rapides pour alimenter le processeur en instructions et en données, et de grande capacité pour stocker l'intégralité des informations disponibles. Or, plus la mémoire et de grande capacité, plus son accès devient lent. Il faut donc utiliser tous les types de mémoires en hiérarchisant et en répartissant les informations en fonction de leur fréquence d'utilisation.

La figure 1 montre comment les mémoires sont organisées selon leur capacité et temps d'accès.

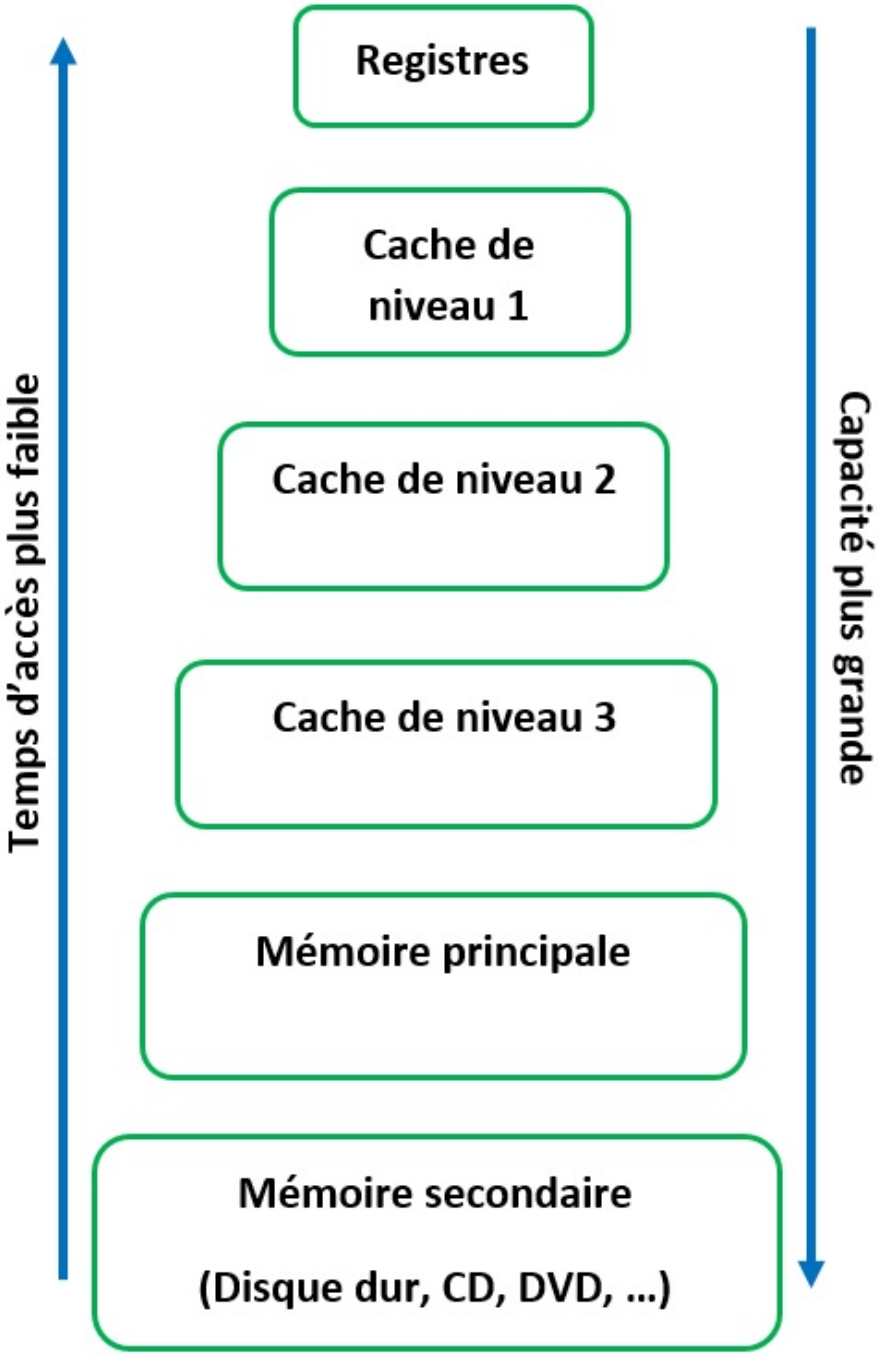

FIGURE 1 – Hiérarchie des mémoires.

Toutes les opérations mémoires de lecture et d'écriture sont dirigées d'abord vers la mémoire cache afin de vérifier la présence ou absence de la donnée. Deux scénarios possibles :

- La donnée est présente : ce cas s'appelle succès ou (hit, en anglais). Dans ce cas, la donnée est transmise au processeur.
- La donnée est absente : ce cas s'appelle défaut ou (miss, en anglais). Dans ce cas, la donnée est transmise d'abord de la mémoire principale vers la mémoire cache pour être délivrée ensuite au processeur.

Un taux de succès est généralement calculé comme la fraction du nombre de succès par rapport au nombre total d'accès au cache.

La mémoire cache stocke de façon transparente les données qui ont été utilisées par le processeur dans un passé proche, dans l'espoir de les réutiliser dans un futur proche. Il en résulte ainsi deux principes de localité :

- **Localité temporelle** : La localité temporelle concerne les données récemment utilisées par le processeur, et qui seront fort probablement réutilisées par processeur. Par conséquent, il faut stocker les données récemment utilisées pour une éventuelle utilisation dans un futur proche. Cela permet de gagner énormément de temps.
- **Localité spatiale** : La localité spatiale stipule que les données voisines de la données actuellement utilisée seront probablement bientôt utilisées par le processeur. Dans ce cas, il est judicieux de charger dans la mémoire cache non seulement les données dont le processeur a besoin, mais également les données voisines. Un bloc mémoire de plusieurs données pourrait être chargé (exemple de 4 mots mémoires).

Pour améliorer le taux de succès du cache, un algorithme s'appelle **Prefetching** qui sera responsable du chargement en avance des données dont le CPU devrait avoir besoin. Les algorithmes de pre-fetching sont basés sur les principes de localité temporelle et spatiale vu précédemment.

3 Caractéristiques de la mémoire cache

Nous avons vu dans la section précédente le principe général de la mémoire cache et comment elle est utilisée par le processeur. Dans cette section, nous allons découvrir les caractéristiques de la mémoire cache en ce qui a traits à la taille, à l'adressage, et aux algorithmes de remplacement.

3.1 Taille d'une mémoire cache

La technologie utilisée dans la mémoire cache met en évidence que plus la mémoire cache est de grande capacité, plus le processeur a de chance d'y trouver de l'information dont il cherche. Par conséquent, pour arriver à des meilleurs performances, on cherche toujours à avoir la mémoire cache avec la plus grande capacité possible.

Cependant, la technologie utilisée dans la mémoire cache fait en sorte que son prix est beaucoup plus élevé que celui de la mémoire principale. Donc, plusieurs

considérations doivent être prises en compte pour arriver à un compromis entre la taille du cache et les performances du processeur.

- La mémoire cache est très rapide mais très chère aussi comparativement à la mémoire principale.
- La mémoire cache occupe plus d'espace si sa capacité est plus grande. La taille d'une mémoire cache varie généralement entre quelques centaines de Ko à quelques Mo. La mémoire cache ne peut donc pas y stocker tout un programme et ses données.
- Le temps d'accès à la mémoire cache est plus rapide si la capacité et faible et vice versa. Donc, cette proportionnalité affecte les performances si la mémoire cache est de grande capacité.

Le constructeur de la mémoire cache doit prendre en compte toutes ces considérations et même d'autres pour arriver à un bonne conception des systèmes.

3.2 Adressage

Comme nous l'avons vu dans la cadre de la mémoire principale, la mémoire cache possède également différents types d'adressage. C'est-à-dire, lorsque une information est transférée de la mémoire principale à la mémoire cache, il faut décider où placer cette information. Il existe principalement trois méthodes pour gérer la correspondance entre une ligne dans le cache et une ligne de la mémoire centrale.

3.2.1 Cache à correspondance direct (ou cache direct)

Le cache direct représente le type d'adressage le plus simple. Dans ce type d'adressage, il y a un emplacement réservé pour chaque ligne mémoire. L'emplacement dans la mémoire cache à déterminé à partir du numéro de ligne mémoire en prenant le numéro de la ligne modulo le nombre de lignes dans le cache. Cela peut être expliqué comme suit :

- Supposons que nous avons C lignes en cache.
- La ligne d'adresse m en mémoire centrale sera gérée par la ligne n en cache avec n =m % C.
- Donc, à partir de l'adresse d'une ligne en mémoire on sait directement dans quelle ligne du cache elle doit se trouver.

Ce type d'adressage a deux avantages importants :

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

Cependant, ce type d'adressage souffre de quelques inconvénients comme par exemple :

- On peut parfois arriver à 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 %.

3.2.2 Cache complètement associatif

Contrairement au cache direct, dans ce type d'adressage, une ligne de la mémoire cache correspond à n'importe quelle ligne de la mémoire centrale comme le montre la figure 2 [1].

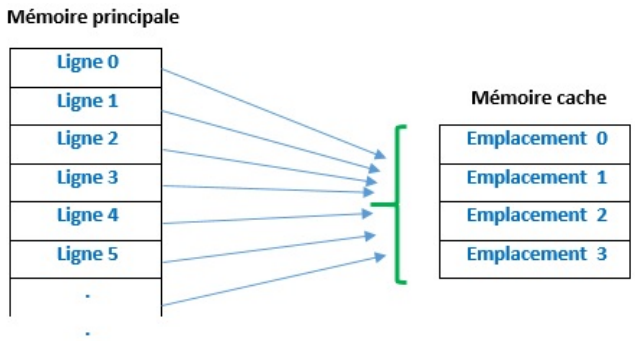

FIGURE 2 – Cache associatif.

L'avantage de l'adressage complètement associatif est qu'il très souple et efficace pour gérer les lignes de manière optimale en terme de succès d'accès. En général, le taux de succès est de l'ordre de 90 à 95%.

Cependant, dans ce type d'adressage, on doit, au pire cas, parcourir toutes les lignes du cache pour savoir si la ligne cherchée s'y trouve ou pas.

3.2.3 Cache mixte ou associatif par ensemble

Ce type d'adressage représente une solution intermédiaire aux 2 solutions précédentes. Dans ce type d'adressage, on regroupe les lignes du cache en ensembles de N lignes comme le montre la figure 3 [2].

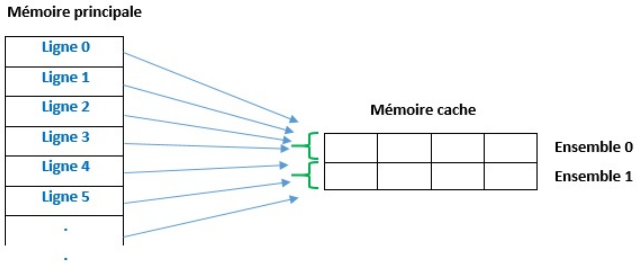

FIGURE 3 – Cache associatif par ensemble de 4, 4 emplacements dans chaque ensemble.

1. Figure tirée du livre d'architecture de l'ordinateur.
2. Figure tirée et modifiée du livre d'architecture de l'ordinateur.

Une ligne de la mémoire centrale est gérée par un ensemble donné donc peut être une de ses N lignes. Donc, une ligne mémoire peut se mettre à n'importe quel emplacement parmi un ensemble précis d'emplacements. Donc, c'est une liberté limitée pour placer une ligne mémoire.

Ce type d'adressage possède beaucoup d'avantages, par exemple :

— Plus souple et efficace que la correspondance directe. Taux de succès de 80% à 90%.
— Temps d'accès très court = 2.5 cycles pour N = 4.
— C'est la méthode utilisée en pratique car elle garantie un meilleur compromis.

3.3 Algorithmes de remplacement

Comme nous l'avons vu dans les sections précédentes, le cache contient une partie des lignes de la mémoire centrale. Si le processeur effectue une opération de lecture ou d'écriture dans une ligne qui n'existe pas dans le cache, à ce moment là, on doit charger en cache cette ligne manquante. Vu que le cache est plein, le processus de chargement des nouvelles lignes dans le cache nécessite d'enlever des lignes pour y mettre d'autres lignes.

Il existe principalement 5 méthodes pour choisir la ligne à remplacer.

— Aléatoire
— FIFO (First In First Out)
— LRU (Least Recently Used)
— LFU (Least Frequently Used)
— NMRU (Non-Most-Recently-Used)

Ces méthodes sont présentées dans les points suivant :

3.3.1 Remplacement aléatoire

Comme son nom l'indique, cet algorithme estime qu'il ne sert à rien de faire des estimations sur les lignes à évacuer. Par conséquent, on se contente de choisir une ligne au hasard pour la remplacer par la nouvelle ligne. Cette ligne pourrait se retrouver dans le cache au complet ou dans un ensemble comme nous l'avons vu précédemment.

Cet algorithme est simple à mettre en oeuvre. Cependant, il est peu efficace en pratique car on peut supprimer des lignes très accédées par le processeur.

3.3.2 Remplacement FIFO

L'idée de base pour cet algorithme est que les informations trop vieilles dans le cache ne vont plus servir et donc elles seront les premières à évacuer lors d'un remplacement.

Chaque ligne du cache contient sa date d'entrée, et la plus vieille est remplacée par une ligne de la mémoire principale.

Cet algorithme est plus efficace que le remplacement aléatoire. Cependant, l'inconvénient de cet algorithme est qu'une ligne référencée régulièrement par

le processeur est retirée automatiquement du cache quand elle devient la plus vieille.

3.3.3 Remplacement LRU

Contrairement au remplacement FIFO, cet algorithme donne de l'importance à l'ancienneté d'utilisation et non pas juste l'ancienneté temporelle. Pour cela, on associe à chaque ligne la date de dernière utilisation et on extrait alors la ligne la plus vieille. Par conséquent, les lignes les plus récemment utilisées ne seront pas évacuées.

Cet algorithme est très efficace, mais il est également très complexe qui requiert la modification de la date à chaque accès.

3.3.4 Remplacement LFU

L'algorithme LFU est considéré comme une variante de l'algorithme LRU. Dans cet algorithme, on associe à chaque ligne un compteur qui s'incrémente à chaque utilisation. Dans ce cas, on extrait la ligne dont le compteur est le plus petit.

L'avantage de cet algorithme est qu'il est plus efficace que l'algorithme LRU, vu que le tri est beaucoup plus simple sur les compteurs que sur les dates.

3.3.5 Remplacement NMRU

Cet algorithme est une variante également de l'algorithme LRU. L'idée de base de cet algorithme est qu'il faut s'assurer que la ligne la plus récente reste toujours dans le cache.

Pour signaler l'utilisation d'une ligne, chaque ligne a un bit supplémentaire qui est mis à 1 lorsque la ligne est utilisée et à 0 sinon. Le remplacement est donc fait aléatoirement parmi les lignes à 0.

3.4 Politique de réécriture

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

Il existe deux méthodes pour la réécriture des données présentées ci-dessous :

3.4.1 Cache write-through

Le principe de cette méthode est simple : Écriture simultanée (write-through). C'est-à-dire, lorsqu'on écrit un mot d'une ligne du cache, on écrit simultanément le mot de la même ligne en mémoire centrale. De cette façon l'écriture est propagée du cache vers la mémoire centrale.

On peut certainement déduire que l'écriture dans ce cas là est plus longue car on effectue un accès à la mémoire centrale.

3.4.2 Cache write-back

Contrairement à la méthode write-through, la méthode write-back (à écriture différée) ne modifie l'information en mémoire principale que lorsque la ligne correspondante disparaît du cache. Cela permet de ne pas perdre trop de temps à accéder à la mémoire principale.

Cependant, il faut toujours penser à réécrire l'information en mémoire lorsque la ligne est évacuée du cache. Pour ce faire, on ajoute à chaque ligne un bit appelé « dirty » (bit sale), qui indique si la ligne a été modifiée et s'il faudra la réécrire en mémoire lors de son évacuation.

4 Amélioration des caches

L'importance des mémoires caches n'a pas cessé d'augmenter avec le temps depuis 1970, date d'introduction du cache. Elle est maintenant une composante indispensable du processeur afin de garantir des performances qui répondent aux besoins. Beaucoup de travaux sont encore en cours pour améliorer davantage les performances des caches selon différents paramètres.

4.1 Temps de récupération d'une instruction

Le temps de récupération peut être calculé en combinant plusieurs paramètres :

— Temps d'accès au cache T_c
— Temps de pénalité au cache T_pénalité
— Taux d'échec du cache T_échec

Le temps moyen pour accéder au cache est calculé comme suit :

$$T\_accès = T\_échec \times T\_pénalité + (1 - T\_échec) \times T\_c \tag{1}$$

Par conséquent, l'amélioration de l'un de ces termes améliore le temps moyen d'exécution et l'efficacité du système en général.

4.2 Causes des défauts de cache

Il existe principalement trois catégories de défauts de cache :

— **Échec obligatoire** : Lors de la première référence à une adresse mémoire, celle-ci ne sera probablement pas dans le cache.
— **Échec de capacité** : Si un programme nécessite de nombreuses données, certaines d'entre elles risquent d'être extraites du cache pour laisser leur place à d'autres.
— **Échec de conflit** : Dans un cache associatif, un emplacement libre peut toujours être occupé par une nouvelle ligne alors que, dans une cache direct ou associatif par ensemble, il peut y avoir un conflit d'accès pour un même emplacement, même si d'autres sont libres.

4.3 Hiérarchiser la mémoire : caches multiniveaux

La mémoire cache est organisée 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'accès aux donnés.
- Cache L1 : généralement scindé en 2 parties Instructions/Données.

Il existe également des relations entre les différents 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.

Les deux types de cache possèdent des avantages et des inconvénients résumés comme suit :
- Inclusif :
  - Cache L2 plus performant.
  - Taille totale plus faible.
  - 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).
- Exclusif :
  - Cache plus grand au total.
  - L2 de taille quelconque.
  - Mais doit gérer la non duplication des données : prend du temps, L2 moins performant.