II. Hiérarchie des mémoires.

Si on devait concevoir une configuration idéale, on concevrait la mémoire comme un ensemble d'un niveau monolithique basé sur une seule technologie. La mémoire résultante ressemblerait à un cache partagé de taille infinie. Malheureusement, le coût économique d'une telle configuration serait exorbitant. Les concepteurs de système doivent donc faire appel à des solutions de compromis et à des hiérarchies de mémoires.

En présence d'une hiérarchie des mémoires, il faut organiser les transferts d'informations entre les différents niveaux, une instruction ne pouvant être exécutée que si elle parvient au processeur. Ces transferts au travers de la hiérarchie des mémoires sont pris en charge soit par le hardware, soit par le système d'exploitation, soit par un composant du système d'exploitation.

1. Généralités.

A. Caractéristiques des mémoires

Les caractéristiques principales des mémoires dont on tient compte dans la conception d'une installation sont :

- Vitesse d'accès
- Bande passante
- Capacité d'extension
- Coût
- Fiabilité
- Paramètres environnementaux : puissance, espace, refroidissement

La vitesse d'accès est importante afin de répondre sans retard à une demande des unités centrales. La bande passante est requise afin de répondre aux demandes de toutes les unités centrales en période de pointe. Le coût, la fiabilité et les paramètres environnementaux sont des facteurs essentiellement économiques.

B. La pyramide

Les diverses technologies à notre disposition nous permettent de mettre au point une structure hiérarchisée de différents niveaux de stockage. Ces différentes technologies dans leur ensemble nous permettent d'atteindre l'objectif principal : obtenir un débit du système dans des conditions économiquement acceptables. Cette hiérarchie utilise un ensemble fort divers de technologies s'étendant de la mémoire vive excessivement rapide (High Speed Buffer) au disque optique.

C. Les composants Hardware

Nous allons discuter dans la suite des différents niveaux caractéristiques d'une hiérarchie typique, nous verrons successivement les composants suivants:

- HSB: High Speed Buffer - la mémoire cache
- Main memory - Random Access Memory (RAM) - la mémoire physique
- Direct Access Devices - Flash, SSD, USB
- Mechanical Hard drives - Disques (DASD)
- Off-line storage - Tapes, Optical Storage

La mise en œuvre de ces composants et les accès aux données peuvent différer d'une architecture à l'autre, ces composants peuvent ne pas correspondre de manière univoque avec un niveau de la hiérarchie. Nous verrons que certains composants hardware sont utilisés à des niveaux différents, la définition du niveau résulte de la combinaison entre le Hardware et le Software soit du µcode (HSB, cache), soit l'O/S (Central, expanded Storage), soit un software spécifique (DASD, K7).

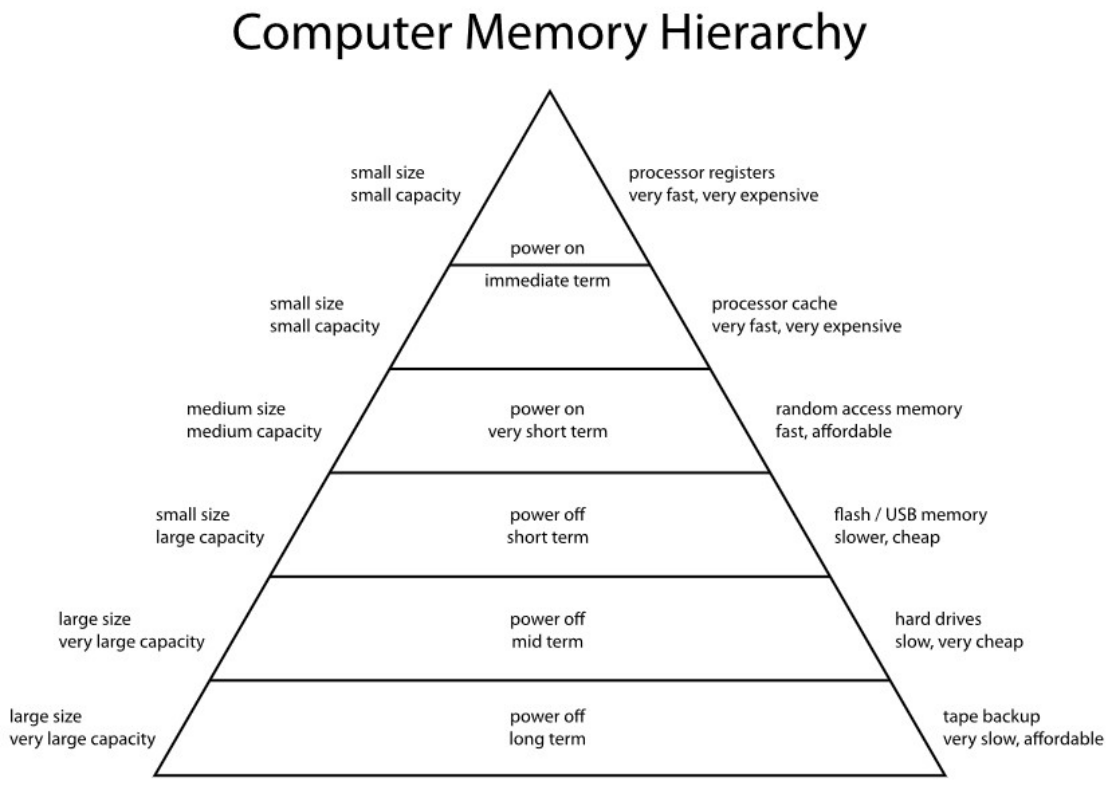


Fig. 7. Composants hardware de la pyramide.

D. Objectif de la hiérarchie

Cet objectif se résume à “ne pas être un frein à la performance des processeurs et ce à un coût économiquement acceptable”.

La hiérarchisation des moyens de stockage est essentiellement motivée par des raisons économiques: plus un moyen de stockage est rapide plus il est cher. Cela se traduit par une relation entre la vitesse d’accès à l’information (de nanosecondes à la minute) et coût par unité de stockage de cette information (du Terabyte ou byte). Ces facteurs en constante évolution sont un des moteurs de l’évolution de l’informatique. Les choix dans cette matière à savoir l’attribution d’un support de stockage à une donnée influencent fortement les performances d’une configuration tant en terme économique qu’en terme de vitesse traitement.

L’objectif final est d’obtenir une vitesse d’accès aussi proche que possible de celle du composant le plus rapide à un coût par unité de stockage équivalent à celui du composant de coût le moins élevé. Cet objectif ne peut être pratiquement réalisé que si la majorité des données sont stockées sur disque et que si la majorité des accès se font directement à partir des caches.

E. Coût Vs Vitesse d’accès

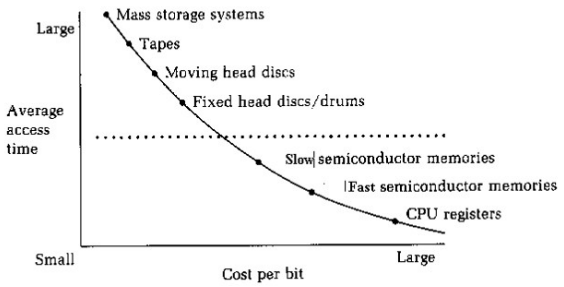


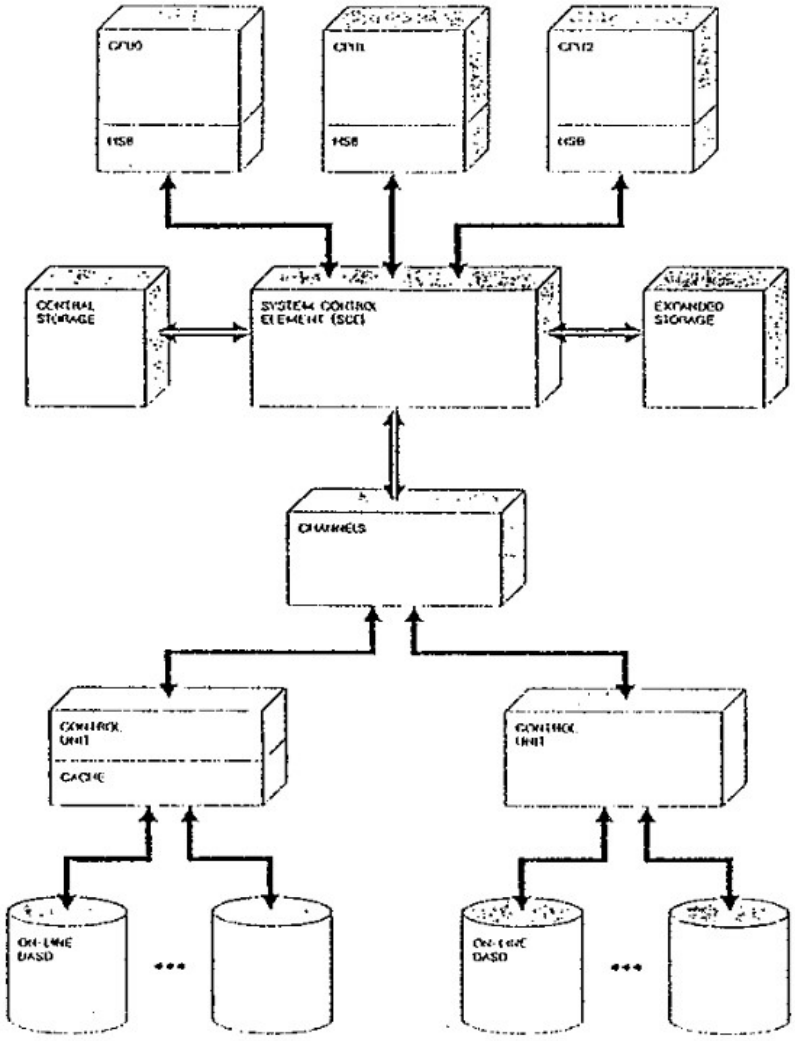
Fig. 8. Evolution des coûts et des vitesses d’accès

Si nous traçons un graphique reprenant l’évolution des facteurs:

- temps d’accès en secondes
- coût par unité de stockage $/B

en fonction de la capacité. Nous pouvons positionner les différents éléments dans la hiérarchie. Lorsque l’on descend dans la hiérarchie, les caractéristiques varient comme illustrées par les courbes de coût et vitesse d’accès.

F. Data Flow



Les données passent d'un niveau à l'autre sur base de la fréquence à laquelle elles sont référencées. Comme nous l'avons déjà indiqué dans la partie du cours consacrée à la gestion de la mémoire, l'étude des accès montre que des accès successifs touchent des morceaux contigus de mémoire, ces morceaux contigus constitue ce qu'on appelle le *working set*. Ces références aux mêmes données permettent d'obtenir un *Hit Ratio* élevé.

Chaque niveau de la mémoire est géré par un composant qui peut être de 2 types:

- hardware ou micro-codé;
- software;

Les stratégies de gestion de ces niveaux sont le plus souvent basées sur l'algorithme LRU. Les éléments de mémoire remplacés descendent d'un niveau dans la hiérarchie. Ces mouvements de données sont appelés suivant les mécanismes ou les systèmes : Migration/recall,
Promotion/demotion,
Staging/destaging, ...

Fig. 9. Flux des données au travers de la hiérarchie.

G. Hit Ratio

Le hit ratio d'un niveau de la hiérarchie est la probabilité (donc exprimée par un nombre compris entre 0 et 1) de trouver une information dans ce niveau de la hiérarchie.

Le hit ratio mesure l'efficacité d'un cache qui finalement représente le pourcentage de références satisfaites par un accès à un niveau de mémoire sans devoir accéder au niveau inférieur. Ce ratio se retrouve à tous les niveaux de la hiérarchie mutatis mutandis. En général, le buffer hit ratio s'étale dans un range allant de 96 à 98 %, des améliorations apportées dans ce range sont d'importance, en effet le nombre de *miss* peut passer alors de 4 à 2 %, soit une diminution de 50 % des attentes du processeur pour des accès à la mémoire.

Il est donc préférable pour rendre les choses plus réalistes de définir le *Miss Ratio* comme étant le pourcentage de références insatisfaites à partir du niveau en question et qui ont donc dû être satisfaites par un accès à un des niveaux inférieurs.

H. Accès Synchrones vs Asynchrones

Le *Processor Storage* comprend l'ensemble des niveaux de la hiérarchie pour lesquels tous les accès du processeur sont synchrones. Donc le processeur attend que le transfert d'un élément soit réalisé à partir de ce niveau de la hiérarchie. Cette partie de la mémoire est fortement intégrée au processeur, le temps d'accès va de la micro-seconde à quelques nanosecondes. Les temps d'accès sont suffisamment bas que pour justifier une pause du processeur. En fait, le processeur ne doit pas attendre si il s'avère que le temps de traitement d'une interruption (i.e. le temps de sauver son état et de rechercher une tâche à activer) est supérieur au temps d'accès à la mémoire. Le *break-even point* est atteint lorsque le temps d'attente du processeur devient égal au temps de recherche et d'activation d'une autre tâche. Un accès est appelé synchrone lorsque le processeur attend que les données lui soient présentées, il devient asynchrone lorsque le processeur travaille à d'autres tâches au lieu d'attendre. Par nature, dans un environnement en multi-programmation, les I/O sont asynchrones et sont traités par une interruption largement évoquées dans le cours de première année.

2. Aspects économiques.

Le coût d'une mémoire est normalement caractérisé en $/bit, ce coût décroît à chaque niveau de la hiérarchie de mémoires, la capacité augmente, mais le temps d'accès augmente également. Le coût moyen d'une hiérarchie des mémoires peut être exprimé par:

$$C_{av} = \frac{c_{main} M_{main} + c_{\sec} M_{\sec}}{M_{main} + M_{\sec}}$$

avec c_{main} et $c_{\sec}$ le coût par bit des mémoires principales et secondaire, M_{main} et $M_{\sec}$ les capacité de ces mémoires. La vitesse d'accès moyenne aux informations va dépendre de la présence de cette information dans les niveaux supérieurs de la hiérarchie. Le temps d'accès est le temps requis entre la requête faite à la mémoire et le transfert effectif de l'information qui s'y trouve. En général, pour une mémoire vive, les temps de lecture et d'écriture sont égaux. Le temps d'accès moyen est donné par l'équation:

$$t_{av} = ht_{main} + (1-h)t_{\sec}$$

ou si toutes les requêtes doivent passer par la mémoire principale:

$$t_{av} = t_{main} + (1-h)t_{\sec}$$

avec t_{main} le temps d'accès à la mémoire principale y compris le temps nécessaire pour traduire les adresses virtuelles en adresses réelles. Nous verrons plus tard les moyens hardware dont nous disposons pour diminuer ce temps. $t_{\sec}$ est le temps d'accès à la mémoire secondaire.

Un critère permettant de chiffrer le coût d'un programme est le produit Espace*temps. Il s'agit de multiplier la quantité de mémoire employée par un programme par le temps durant lequel cette mémoire est mobilisée par le programme. Puisque ces deux éléments sont directement reliés au coût, ce produit est considéré comme une indication du coût d'exécution d'un programme. L'utilisation de la mémoire par un programme variant en fonction du temps, ce produit s'exprime par l'intégrale du nombre de pages employées sur l'intervalle de temps d'exécution en ce compris le temps d'attente pour transférer l'information de la mémoire secondaire

$$ST(t_1, t_2) = \int_{t_1}^{t_2} M(t)dt$$

avec M(t) le nombre de pages de mémoire mobilisées à l'instant t.

Pour un système à allocation de mémoire fixe, le produit ST se réduit à

$$ST = M(nt_{main} + ft_{\sec})$$

avec n le nombre de références sur l'intervalle de temps, et f le nombre de pages faults. L'équivalence entre les pages faults et le miss ratio, nous amène à l'expression du produit St pour un système à allocation fixe:

$$ST = Mnt_{av}$$

le coût d'un programme augmentera donc en fonction de son nombre de références et diminuera en fonction du hit ratio dont le temps d'accès moyen dépend.

3. High Speed Buffer – mémoire cache

A. Description.

Ce composant est situé au sommet de la hiérarchie sous les unités centrales et sert en fait de cache pour ces unités. C'est l'élément le plus critique car toutes les références des unités centrales aussi bien à des instructions qu'à des données se font à partir de ce composant. Ses performances ont donc un impact direct sur les performances des unités centrales. Le temps d'accès de ces mémoires doit donc est fort proche de la vitesse des unités centrales sous peine de ne pouvoir exploiter les possibilités de ces dernières. Cette même contrainte impose la proximité entre ces mémoires et les unités centrales afin de minimiser les retards dus à la propagation des signaux. Les CPU modernes ont différents niveaux de cache : tout d'abord le cache de niveau 1 (de type SRAM et de latence de l'ordre des 10ns) est souvent divisé en une « L1 cache » pour les données et une autre pour les instructions, toutes deux spécifiques à chaque cœur du CPU. Un cache de niveau 2 (de type DRAM), plus grand mais plus lent (de l'ordre des 100 ns), est généralement partagé entre différents cœurs, le cache de niveau 3 étant souvent partagé entre tous les cœurs.

La capacité de ce composant dépasse rarement quelques centaines de KiloBytes jusqu'à quelques MegaBytes, suite aux limitations d'espace mais surtout de coût. Cette capacité n'est pas un inconvénient si la localisation des accès est élevée. Lorsqu'une donnée n'est pas trouvée dans le cache, elle doit être retrouvée à partir du niveau inférieur c.à.d. le composant *Central Storage*. Ces accès sont plus lents mais toujours synchrones c.à.d. n'entraînant pas d'interruption, le processeur doit donc attendre le transfert de données. Ce transfert est exécuté en parallèle vers le cache et vers le processeur.

Le High Speed buffer est un cache, son efficacité pourrait être limitée suite à sa taille réduite. Il n'est en rien suite au principe de localisation des références qui semble s'appliquer de manière empirique au comportement des programmes, tant aux données qu'aux instructions. On distingue deux types de localisation:

- Localisation temporelle: une adresse mémoire une fois référencée a de grandes chances de l'être à nouveau dans un futur proche.
- Localisation spatiale: la prochaine référence mémoire a de fortes chances d'être fort proche de la précédente.

La localisation temporelle peut être la conséquence de boucles, de l'emploi de piles (stacks)… dans la programmation. La localisation spatiale résulte de la construction des programmes: emploi de procédures, localisation séquentielle.

Ces deux principes tendent à faire admettre qu'un grand nombre de références à la mémoire peuvent être satisfaites en mobilisant fort peu de capacité mémoire cache.

Notons pour la petite histoire, que la vitesse de traitement d'un ordinateur va dépendre fatalement des possibilités du hardware dont il est constitué mais également du type de sollicitations. Afin de mesurer les possibilités une machine, il est essentiel de lui présenter une charge de travail qui aura été étalonnée auparavant. Imaginons que les possibilités d'une machine A soient mesurées en la sollicitant par un programme dont

le comportement est purement séquentiel, et une machine B en la sollicitant par un programme constitué d'une petite boucle, la machine B même équipée d'une unité centrale plus lente pourrait obtenir un débit (en millions d'instructions par secondes) plus élevé que la machine A.

Le principe de localisation permet à un cache d'augmenter nettement la vitesse de traitement d'un système.

B. Hit Ratio.

La probabilité de trouver une information dans le cache dépend du comportement d'un programme mais aussi de la taille et de l'organisation du cache. Typiquement, 80 à 90% des références vont trouver l'information dans le cache. On appelle « hit » le fait de trouver une information directement dans le cache, à l'inverse, on appelle « miss » le fait de ne rien trouver dans le cache et donc de devoir accéder un niveau inférieur de la hiérarchie. Le hit ratio d'un cache s'exprime comme pour tout niveau de la hiérarchie comme :

h= Nombre de références satisfaites par le cache / Nombre total de références

Supposons que le temps d'accès à la mémoire centrale d'un ordinateur soit de 100 nanosecondes et que l'unité centrale soit capable d'exécuter 50 millions d'instructions en une seconde :

$$50\ 10^{6}\ *\ 100\ 10^{-9}\ =\ 5$$

chaque accès à la mémoire centrale coûterait 5 instructions au processeur avant de recevoir ses données.

Pour illustrer l'importance du *Miss ratio*, reprenons le même ordinateur et associons un cache dont le temps d'accès est 25 nanosecondes.

Un Hit Ratio de 96% soit un *Miss Ratio* de 4% signifie que 4 accès sur 100 ont un temps de réponse de 100 nanosecondes soit 5*4 instructions non déroulées. 96 accès comptant pour 1.25 instructions non déroulées. Au total, nous atteignons 96*.25 +4*5= 44 instructions non déroulées. 98% de Hit Ratio soit 2% de *miss ratio* nous coûterait 98*0.25+2*5= 34.5 instructions soit une amélioration de 21 %.

Dans une configuration pareille, le processeur est capable de traiter 50 millions d'accès par seconde, une mémoire cache dont le temps d'accès est de 50 nanosecondes, ne permettrait à ce processeur d'atteindre moins de la moitié de sa vitesse théorique soit 20 millions d'instructions.

Nous avons vu que le temps d'accès moyen dans un système à 2 niveaux de mémoire s'exprimait par

$$t_{av} = t_{mem1} + (1-h)t_{mem2}$$

avec t_{mem1} le temps d'accès au premier niveau de mémoire (cache dans un système avec cache, mémoire virtuelle principale dans un système à mémoire virtuelle), t_{mem2} le temps d'accès au second niveau (mémoire centrale dans un système avec cache, mémoire secondaire dans un système à mémoire virtuelle). (1-h) représente le miss ratio.

Cette expression devient

$$t_{av} = t_{mem2}(1/k + (1-h))$$

avec k le ratio entre les temps d'accès. Nous déduisons de cette expression que le temps d'accès moyen va être dominé par le miss ratio si le facteur k est élevé. En supposant que nous travaillons avec des disques dont le temps d'accès est de 20 millisecondes et une mémoire RAM dont le temps d'accès est de 200 nanosecondes, le facteur k vaut 100.000. Dans l'exemple traitant du cache, ce facteur valait 4. Pour ce dernier type de niveaux, le miss ratio ne doit pas être fort bas pour que le temps d'accès moyen s'approche du temps d'accès du cache.

C. Gestion du cache en lecture.

Il existe 3 types de stratégies adoptées dans la remontée d'informations de la mémoire centrale vers le cache:

- Demand fetch
- Prefetch
- Selective fetch

Les deux premières stratégies ont déjà été évoquées dans la partie du cours de première année qui traite des algorithmes de pagination. Demand fetch consiste à remonter les blocs de la mémoire centrale vers le cache en réponse à une demande. Prefetch consiste à remonter les blocs avant que ceux-ci ne soient demandés. Le plus simple consiste à remonter le bloc i+1 lorsque l'on référence le bloc i. Cette stratégie peut être coûteuse en espace dans la cache, le "prefetch on a miss" consiste à remonter le bloc i+1 en cas de miss sur le bloc i.

Selective fetch consiste à ne pas systématiquement remonter des informations dans le cache. Par exemple, des variables partagées, qu'il est plus simple pour des raisons d'intégrité de conserver dans la mémoire centrale spécialement dans un système multiprocesseurs. Des zones mémoire contenant uniquement des données pourraient être marquées comme étant à conserver en mémoire centrale car susceptibles d'être modifiées, alors que des zones mémoire contenant des instructions non sujettes à modification seront recopiées dans le cache.

D. Gestion du cache en écriture.

La lecture d'une information dans la cache n'altère pas le contenu du cache, il n'y a donc aucune différence entre l'information contenue dans le cache et sa copie contenue en mémoire centrale. L'écriture dans le cache va par contre entraîner l'apparition de différences entre cache et mémoire centrale. Il devient donc nécessaire de modifier à la fois le cache et la mémoire centrale si d'autres processeurs (I/O par exemple) accèdent à la mémoire centrale. Le temps d'accès moyen incluant les opérations d'écriture va dépendre des mécanismes employés pour garantir la cohérence des données. Les deux stratégies les plus couramment employés sont:

- write-through (tru en américain)
- write-back

a) Write-tru.

Chaque opération d'écriture est directement répétée vers la mémoire centrale. L'opération d'écriture en mémoire central va donc prendre plus de temps et va dominer le temps d'accès pour toutes les opérations d'écriture. Le comportement d'un programme normal nous indique que le nombre d'écritures représente entre 5 et 34 pour-cent des lectures ce qui atténue fortement la dégradation de performances due à ce mécanisme qui est aussi appelé Store-tru;

Le temps d'accès moyen peut s'exprimer par

$$t_{av} = (1-w)t_{mem1} + (1-w)(1-h)t_{mem2} + wt_{mem2}$$

avec w le pourcentage d'opérations d'écriture. Cette expression est basée sur l'hypothèse que lors d'un miss sur une opération d'écriture, on ne remonte pas les informations dans le cache.

Les opérations d'écriture en mémoire centrale peuvent être accélérées en employant des buffers intermédiaires comme illustré dans le schéma ci-dessous.

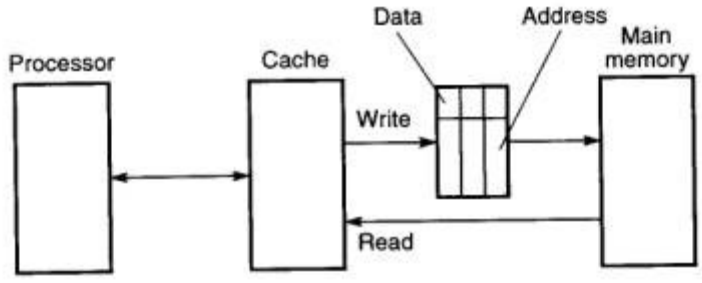


Fig. 10. Cache et buffers intermédiaires.

Ces buffers contiennent les données modifiées et leur adresse, il s'ensuit que toute référence à la mémoire doit nécessairement vérifier la présence de données dans ces buffers par comparaison des adresses. L'unité centrale peut continuer les opérations

alors que la mémoire centrale est mise à jour à sa vitesse, le nombre de buffers requis est d'autant plus restreint que les facteurs k et w sont faibles.

b) Write-back.

Ce mécanisme implique qu'un bloc du cache est écrit en mémoire centrale s'il est remplacé dans le cache (flush). Le mécanisme le plus simple consiste à recopier à chaque fois le bloc qu'il ait été modifié ou non. Dans ce cas, le temps d'accès moyen s'exprime:

$$t_{av} = t_{mem1} + (1-h)t_{mem2} + (1-h)t_{mem2}$$

Un mécanisme plus élaboré implique l'écriture uniquement des blocs qui ont été modifiés ; le temps d'accès moyen s'exprime alors:

$$t_{av} = t_{mem1} + (1-h)t_{mem2} + w(1-h)t_{mem2}$$

c) Multiprocesseurs.

Le mécanisme basé sur le write-tru n'est pas suffisant pour maintenir une cohérence complète des caches dans un environnement multiprocesseurs.

Différentes techniques peuvent être envisagées:

- cache partagé
- éléments « non-cachables »
- snoop bus
- diffusion de messages
- emploi d'une directory

L'emploi d'un cache partagé répond à la contrainte de cohérence des données, un DMA pourrait même y avoir accès. Cependant, les performances de ce type de mécanisme se dégradent fort avec l'augmentation du nombre de processeurs.

Les problèmes de cohérence de cache surviennent en cas de modification d'une variable commune, ces données peuvent être définies comme non-cachables. Leur modification ne peut intervenir qu'au sein d'une section critique basée sur des mécanismes software dont nous verrons le détail dans le chapitre consacré à l'IPC (Inter-Process Communication).

Le mécanisme dit Snoop Bus est basé sur une surveillance du trafic sur le bus, il s'agit donc d'un mécanisme couramment employé au sein des microprocesseurs. Un composant surveille le trafic sur le bus et détecte en particulier les opérations d'écriture . Si une adresse mémoire présente dans un des caches est modifiée, ce composant invalide l'entrée dans le cache. Ce mécanisme est illustré à la figure ci-dessous (80386), il emploie fatalement la technique write-tru.

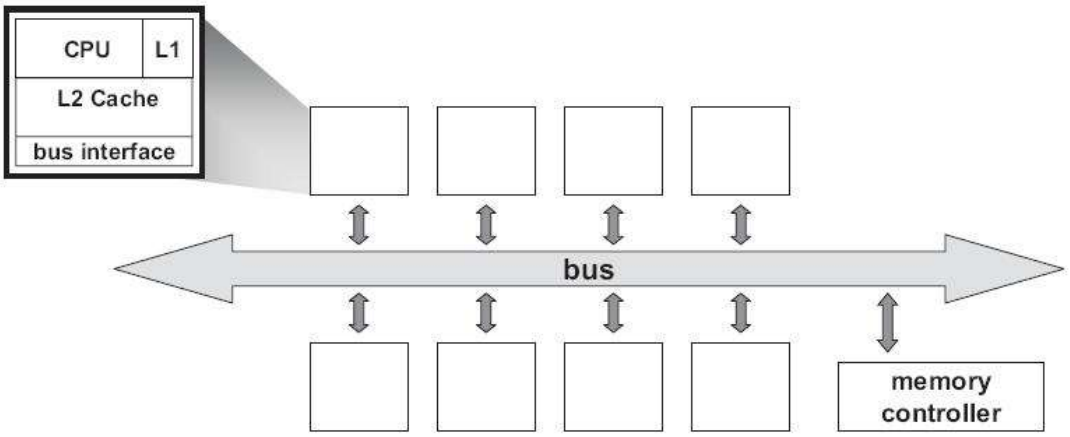


Fig. 11. Cache et Snoop bus controller

La diffusion de messages repose sur la signalisation à tous les caches de la modification d'une information dans l'un d'entre eux. La donnée incohérente dans un des caches est soit invalidée soit rafraîchie. Cette technique (invalidation) est utilisée dans les gros ordinateurs.

Une directory peut être adjointe à la technique d'invalidation ; cette technique repose sur le principe suivant: si un bloc n'a pas été modifié, il peut se trouver de manière simultanée dans plusieurs caches. Une fois ce bloc modifié dans un des caches, les copies incohérentes sont invalidées. L'information ne sera rafraîchie qu'en cas de tentative de lecture. Cette méthode emploie une directory, constituée d'un ensemble de bits pour chaque bloc pouvant être transféré dans un cache. Chaque bit de cet ensemble attaché à un bloc correspond à un cache dans lequel le bloc est potentiellement présent. Un bit supplémentaire est employé pour indiquer que le bloc a été modifié. La directory doit permettre de répondre aux situations suivantes:

- read hit: si le bloc est présent dans le cache aucune opération n'est nécessaire ;
- read miss: si un accès mémoire est nécessaire, il faut vérifier si le bit indiquant une modification est mis « on », auquel cas une copie fraîche doit être transférée et les autres copies invalidées. Le bit indiquant une modification subit également un « reset ».
- write hit: le set de bits est vérifié afin de savoir si d'autres copies de l'information sont présentes dans les autres caches. Si tel est le cas, ces copies sont invalidées...

et ainsi de suite.

4. Processor Storage

A. Description.

Le processor storage d'un système supportant l'adressage virtuel est le plus souvent constituée de deux niveaux :

- la mémoire centrale (main memory, central storage,..)
- la mémoire étendue (Extended memory, expanded memory,...)

La mémoire centrale est une mémoire rapide, adressable au niveau du byte ce qui signifie qu'une adresse générée par un processeur indique l'adresse d'un byte de donnée. Comme expliqué précédemment, le hardware maintient pour chacun des frames une clé de protection, un indicateur de changement et un indicateur de référence. Ces indicateurs sont stockés dans une mémoire excessivement rapide de sorte que l'algorithme de pagination du système d'exploitation puisse les accéder rapidement.

Le hardware assure également une protection contre la perte de données

- correction d'une erreur simple;
- détection d'une erreur double;

En fonction des architectures, les transferts de données se font de la manière suivante:

- de la mémoire centrale vers le cache par fraction de frame (128 bytes par exemple) ; temps d'accès de l'ordre de 100 nanosecondes. Ces transferts sont organisés par le microcode ;
- de la mémoire étendue vers la mémoire centrale par frame ou ensemble de frames ; temps d'accès de l'ordre de la microseconde;
- des unités externes (disque, contrôleur TP, ...) vers la mémoire étendue ou vers la mémoire centrale en fonction de l'architecture ; temps d'accès de l'ordre de la milliseconde. Quantité de 80 à 32K ou plus suivant les architectures. Ces deux derniers transferts sont organisés par le système d'exploitation dans le cadre de la pagination.

La mémoire étendue est une mémoire Frame, adressable en d'autres termes ; une adresse générée par un processeur indique l'emplacement d'un frame le plus souvent de 4K. La mémoire étendue dispose des mêmes mécanismes permettant d'assurer l'intégrité des données et les fonctions de gestion de la mémoire que la mémoire centrale. Cependant, ces éléments sont en général gérés par le software (Système d'exploitation) et non par un hardware/microcode dédicacé. Le niveau de protection est plus élevé, cette mémoire est moins rapide, moins chère mais aussi moins fiable: l'*overhead* dû à la correction est moins significatif. Les mécanismes employés sont:

- correction d'erreur double;
- détection d'erreur triple;

La mémoire centrale et la mémoire étendue sont gérées par un algorithme de remplacement des pages dont nous verrons ultérieurement le fonctionnement. Une page peut rester non référencée plusieurs dizaines de secondes en mémoire étendue avant de se retrouver dans un niveau inférieur de la hiérarchie.

B. Calcul des temps d'accès.

Nous avons expliqué le fonctionnement de trois niveaux de la hiérarchie présents dans un système supportant les mémoires virtuelles. Nous verrons plus loin dans ce cours qu'un dispositif hardware est employé pour accélérer le mécanisme de traduction dynamique des adresses virtuelles en adresses réelles (en abrégé DAT, pour « Dynamic Address Translation »). Ce dispositif s'appelle le « Translation Lookaside Buffer » (en abrégé TLB). Il se comporte comme un cache contenant les tables de pages et est donc accédé lors de chaque accès mémoire.

Connaissant ces éléments, nous sommes capables de calculer les temps d'accès moyen d'un système comportant un cache, et supportant une ou plusieurs mémoires virtuelles paginées.

Six situations peuvent survenir dans ce type de système.

1. le DAT emploie les informations présentes dans le TLB et les données se trouvent dans la cache.
2. le DAT emploie les informations présentes dans le TLB et les données se trouvent en mémoire centrale.
3. le DAT emploie les informations présentes dans le cache, les données sont présentes dans le cache
4. le DAT emploie les informations présentes dans le cache, les données sont présentes en mémoire centrale.
5. le DAT emploie les informations présentes en mémoire centrale, les données sont présentes en cache
6. le DAT emploie les informations présentes en mémoire centrale, les données sont présentes en mémoire centrale.

Supposons que les temps d'accès suivants sont d'application:

Temps d'accès au TLB = 25 ns
Temps d'accès au cache = 25 ns
Temps d'accès à la mémoire centrale = 200 ns
Hit ratio TLB = 90%
Hit ratio cache = 95%

Les calculs de temps d'accès et de probabilité d'apparition des différentes combinaisons sont synthétisées dans la tableau suivant:

Combinaison	Accès TLB	Accès HSB Adresse	Accès HSB Data	Accès mémoire Adresse	Accès mémoire Data	Total		Probabilité	Temps Moyen
1	25		25			50	0.9x.095	0.85500	42.75
2	25		25		200	250	0.9x0.05	0.04500	11.25
3	25	25	25			75	0.1x0.95x0.95	0.09025	6.76875
4	25	25	25		200	275	0.1x0.95x0.05	0.00475	1.30625
5	25	25	25	200		275	0.1x0.05x0.95	0.00475	1.30625
6	25	25	25	200	200	475	0.1x0.05x0.05	0.00025	0.11875
									63.5

Nous constatons que le temps d'accès moyen vaut 63,5 nanosecondes.

5. I/O Boundary

Le niveau suivant le processor storage est le plus souvent constitué par les disques magnétiques. Ces disques sont basés sur des dispositifs mécaniques (actuator) constitués d'un ensemble de têtes magnéto-résistive (la résistance change en fonction de l'intensité d'un champ magnétique) attachées à un bras se déplaçant au-dessus de plateaux tournant à une certaine vitesse.

Toutefois, depuis quelques années s'est démocratisé un niveau intermédiaire entre la mémoire RAM et les disques magnétiques : le SSD, pour Solid State Drive.

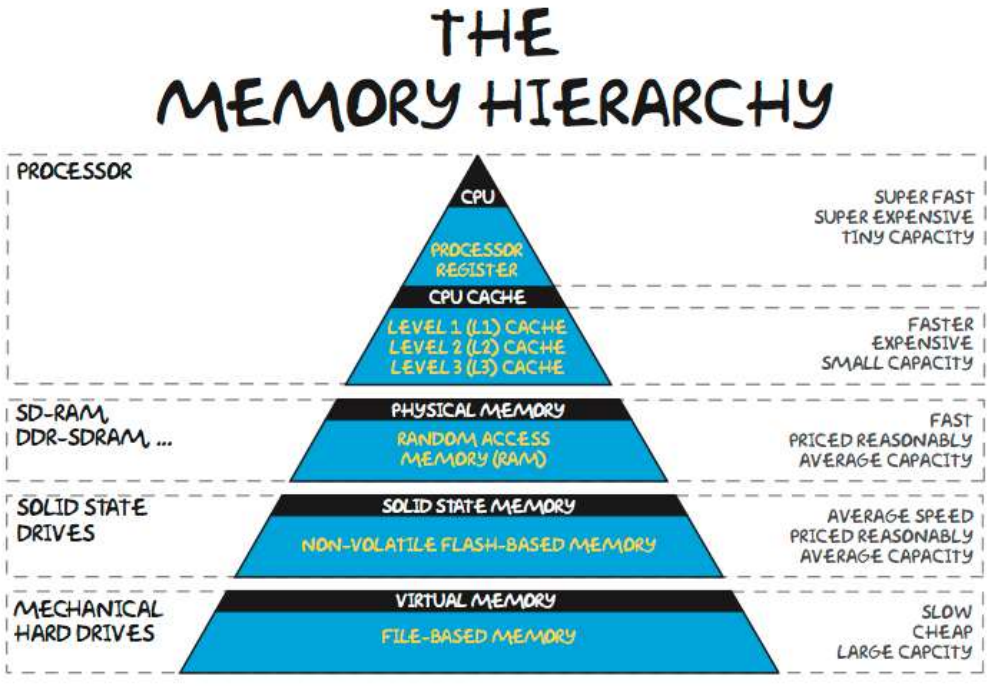


Fig. 12. Hiérarchie de mémoire.

Un SSD est basé sur de la mémoire flash, qui a la caractéristique de conserver les données lors d'une mise hors tension, contrairement à la mémoire RAM. Un SSD ne contient pas d'éléments mécaniques, et est donc moins fragile et plus silencieux que les disques durs classiques. Les SSD ont des performances largement supérieures aux disques durs : temps d'accès de 0.1 ms à comparer à environ 10 ms, débit de lecture

supérieur, mais moins en écriture (donc convient particulièrement bien pour héberger le système d'exploitation et démarrer rapidement). Les disques SSD restent plus chers que les disques durs, c'est pourquoi on trouve par exemple maintenant souvent dans les PC un SSD de max. quelques centaines de GB associé à un disque dur de 1 TB ou plus.

SSD ou disques magnétiques classiques (nous entrons ici dans le domaine mécanique), au contraire de ce qui se passe lors d'accès mémoire, les accès disques justifient amplement le traitement d'une interruption. Nous venons de traverser l'I/O boundary, en d'autres termes les accès du processeur aux données deviennent asynchrones.

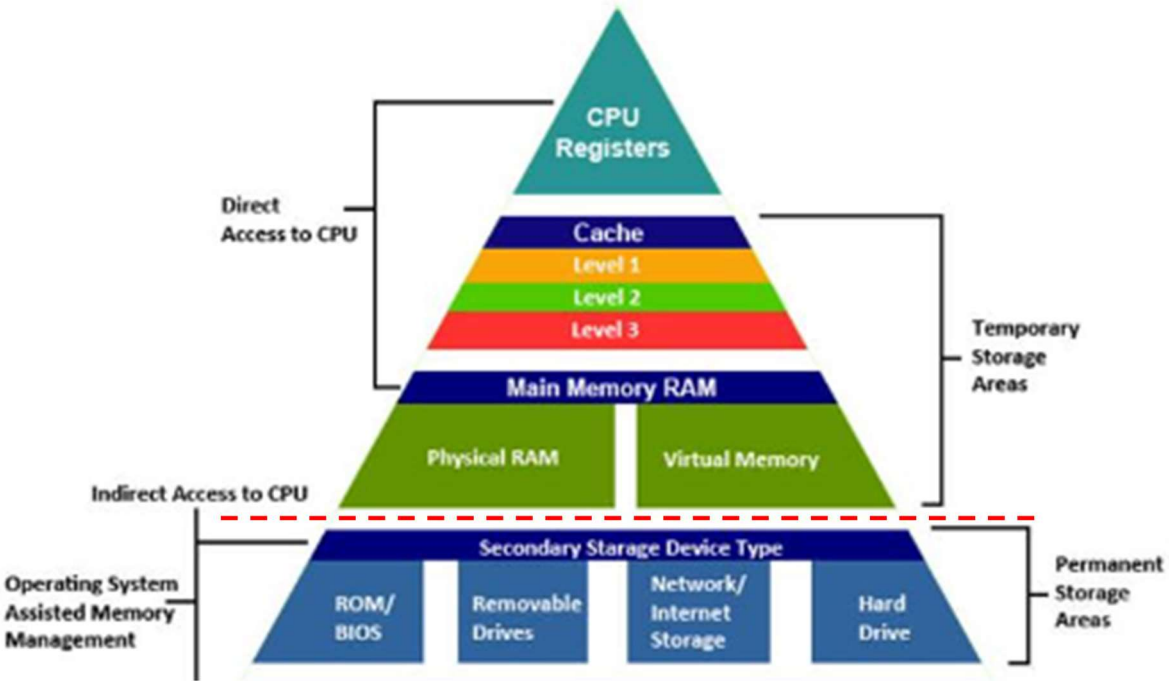


Fig. 13. Hiérarchie de mémoire et I/O boundary.

A. Les unités E/S à accès direct.

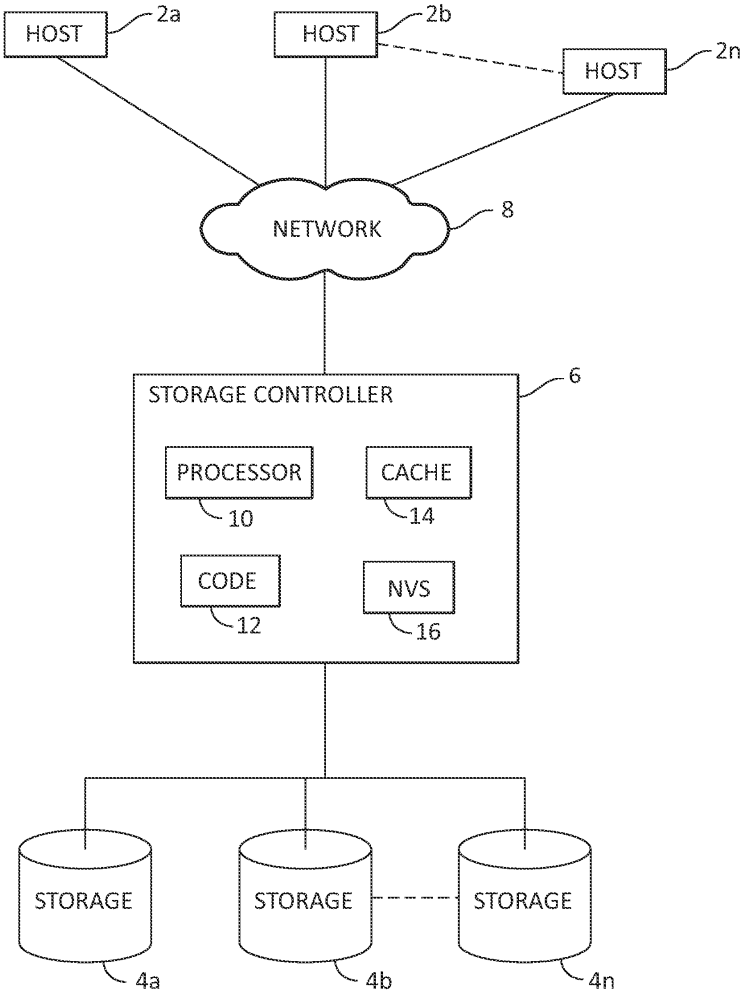


Fig. 14. Unité de contrôle de disques.

On distingue 2 types :

- Block devices: disques, un bloc est adressable directement
- Character devices : un stream de caractères est transmis

Structure standard :

- un contrôleur ou unité de contrôle (Control Unit).
- des disques (32,64,128?..) appelés DASD pour Direct Access Storage Device

Le but de la CU est de traduire les ordres du processeur en ordres de positionnement des éléments mécaniques. De plus, la CU gère la communication au travers du channel.

Le flux des données est le suivant :

1. un processus tente d'accéder une donnée;

2. cette donnée ne se trouve dans aucun des niveaux de la hiérarchie synchrone;
3. l'O/S (partie méthode d'accès) rédige une requête d'entrée/sortie (ordre de positionnement, endroit de la mémoire où placer les infos ...) et interrompt le processus en cours;
4. cette requête I/O est interprétée par l'I/O supervisor qui transmet les requêtes successives aux disques via le channel;
5. la requête de positionnement est traduite (analogique/digital) et transmise au device;
6. une fois positionné, le disque transmet les informations (connect time);
7. ces informations sont placées dans le cache de la CU et transmises au processeur via le channel;
8. le processeur reçoit une interruption lorsque le bloc complet est transmis. Principe des DMA (Direct Memory access CU). Sinon, le bus est occupé à chaque transfert d'un mot, le CPU boucle pour lire ce mot ...

B. Internal design DASD

La structure interne d'un disque normal (HDA pour Head Disk Assembly) reprend les éléments ou concepts suivants :

- têtes de lecture placées sur un bras articulé
- disques support du composant magnétique (thin film)
- chaque disque = une piste
- chaque piste séparée en secteurs
- chaque disque partagé en cylindres
- mouvement du bras: seek time
- mouvement de rotation: latency.

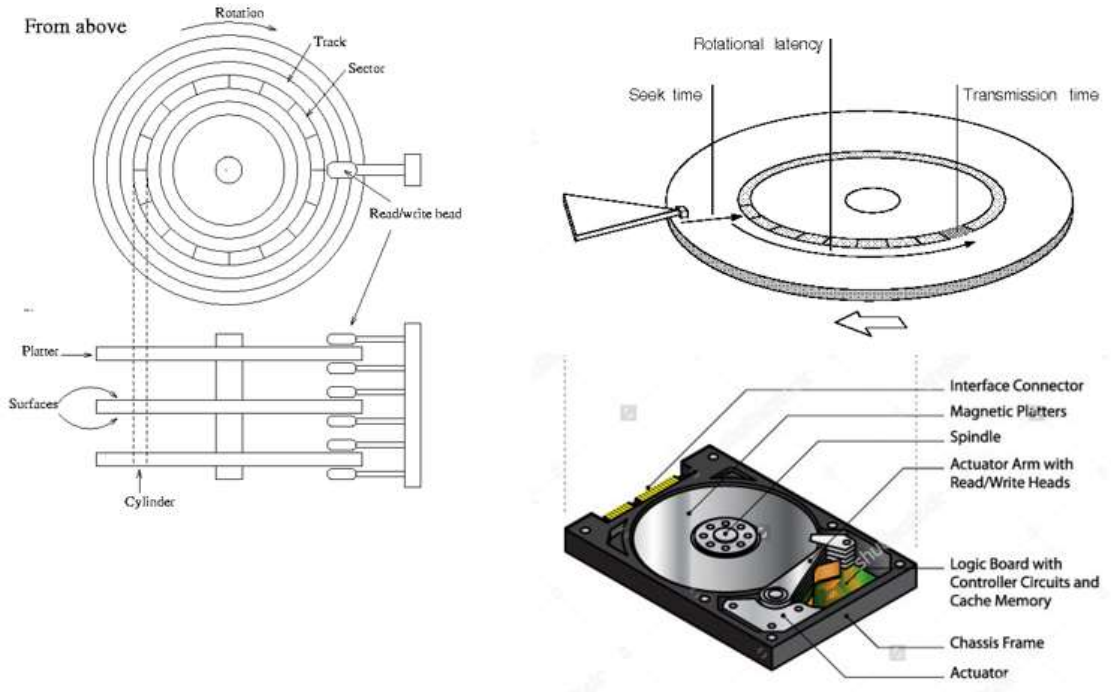


Fig. 15. Conception interne d'un disque.

La tête magnéto-résistive est constituée d'un matériau dont la résistance varie en fonction du champ magnétique. Si on lui applique une différence de potentiel, on mesure l'intensité du courant qui varie en fonction de la résistance, donc du champ magnétique (I=U/R).

C. Algorithmes de gestion des caches.

Le cache disque est une mémoire RAM placée entre le disque et la mémoire centrale d'un système. La taille des caches actuels varie entre 8 Mbytes et 2 Gbytes, ils contiennent des blocs de données qui suivant les principes de localisation des références ont toutes les chances d'être utilisés dans un futur proche, ce qui permet d'atteindre des temps d'accès moyen compris entre 2 et 5 millisecondes contre 20 à 25 millisecondes en l'absence de cache.

Certains systèmes d'exploitation comme UNIX emploient la mémoire centrale d'un système comme cache disque. Cependant, il est plus fréquent de trouver la mémoire cache au sein de l'unité de contrôle afin d'apporter un bénéfice direct sans modification d'un système existant. En cas d'écriture, la mécanique write-tru est employée dans la but de simplifier le recovery. Les unités de contrôle moderne disposent d'un NVS (Non Volatile Storage) qui est conservé même en cas de panne de courant. Le NVS contient la copie des informations qui ont été modifiées dans le cache tant que ces dernières n'ont pas été écrites sur disques.

a) Cache: Read Hit

Un read hit signifie qu'une opération de lecture peut être satisfaite à partir du cache. Aucun accès physique au disque n'est nécessaire.

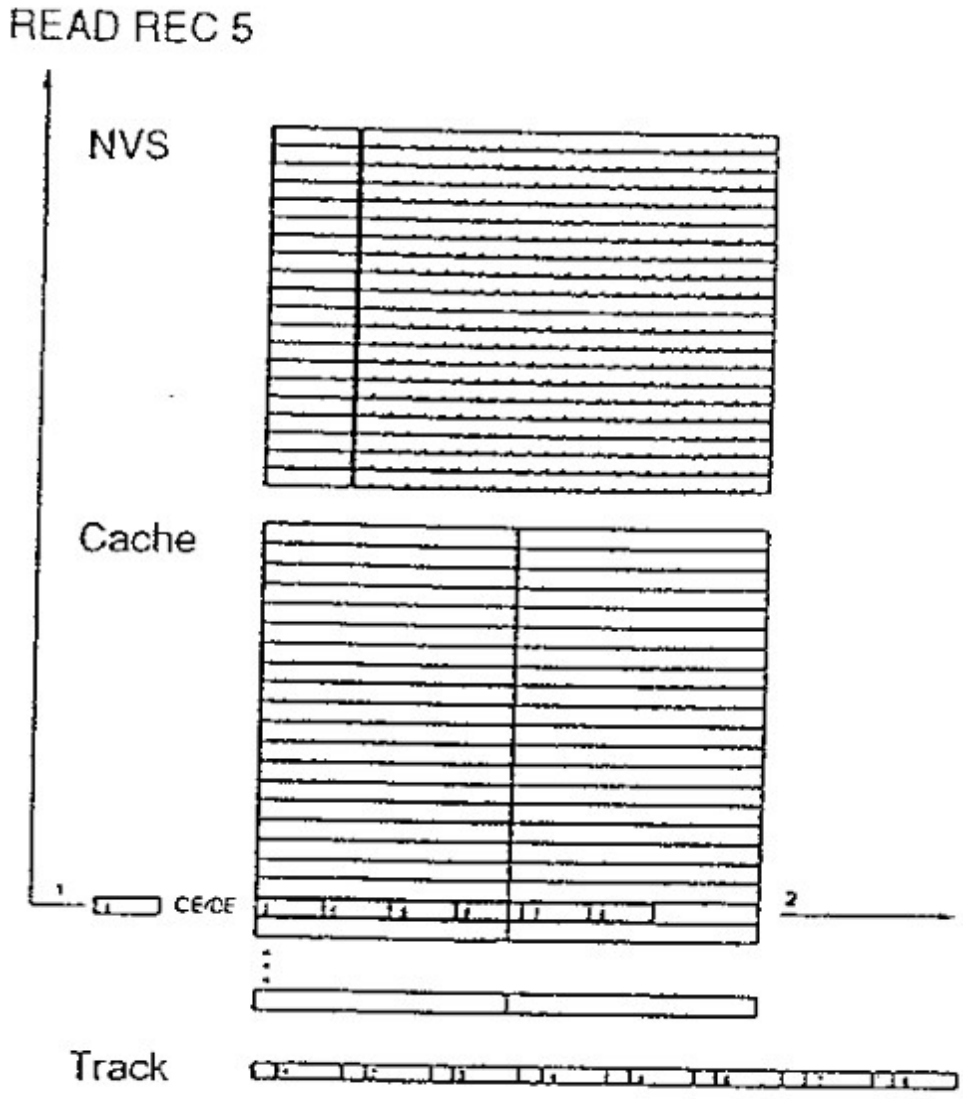


Fig. 16. Cache read hit.

b) Cache : staging

Une référence sur un disque s'appelle comme pour tous les niveaux rencontrés un *read miss*. Suivant les algorithmes de gestion du cache, nous pouvons remonter dans le cache:

- une piste complète;
- la piste à partir du record jusqu'à la fin de la piste;
- le record;
- un ensemble de pistes contiguës.

c) Cache : Write Hit

Un write hit consiste en une écriture réussie dans le cache et dans le NVS (Non Volatile Storage) ; il s'agit d'un cache spécial dont les caractéristiques sont :

- Record managed.
- Directory
- Records Image (no locality)
- Free pointer, clean-up pointer

La gestion du NVS se rapproche du problème du consommateur et du producteur qui sera développé dans le cadre de la communication inter-processus. Un processus produit des records, il utilise le pointeur des records libres, un processus consomme les records, il utilise le clean-up pointer. Le premier est mis en attente si le buffer est plein, le second est mis en attente si le buffer est vide. Ils ne peuvent travailler simultanément sur le même record.

d) Cache residency.

Le cache residency time représente le temps durant lequel une information (bloc, piste, slot,..) réside dans le cache. Ce temps dépend de l'activité et de la taille du cache, la relation entre ces éléments s'exprime comme suit:

$$S = T * (1-h) * r$$

Avec

S = la taille du cache en slots,records,..
T = le residency time
h = le hit ratio
r = le nombre d'opérations E/S par intervalle de temps

Taille d'une population = Residency time*arrival rate.
Comparaison pour la population d'un pays : sa taille dépend de l'âge moyen et de la rapidité de reproduction.
Ex : Les chinois sont 1.4 milliard, et vivent en moyenne 75 ans.
1.400.000.000 = 75*365*24*3600*Arrival rate
→ arrival rate = 0.6 par seconde
Pour que la population chinoise ne croisse pas, il ne faut pas plus d'une naissance toutes les 1/0.6 = 1.7 sec.

Un raisonnement équivalent s'applique à la gestion du cache mutatis mutandis:

Le Hit ratio = (Nbre de références - Nbr de staging) / Nbre de références
donc Nbr de staging = (1-Hit ratio) * Nbr de références

Illustrons cette formule par un exemple chiffré, supposons que

r = 70 E/S sec
S = 336 slots
h = 85 %

Alors T = 336 / (1-0.85) *70 = 32 secondes.

e) Hypothèse de Mc Nutt

Si un disque partage le cache avec d'autres, le *hit ratio* du disque peut être exprimé en fonction du *Residency time* du cache entier.

Illustrons cette hypothèse par un exemple chiffré, supposons que

ri = 10 E/S sec
hi = 95 %
T = 32 secondes
Donc S = 32 * (1-0.95) * 10 = 16 slots
Si ri=20 et hi=50%, S = 32 * (1-0.5) * 20 = 320 slots
Si ri=1 et hi=100%, S = 32 *(1-1) *1 = 0 slots

Pour éviter ces dysfonctionnements, on met en œuvre des politiques de gestion basée sur LRU et le sequential caching (prefetch).

6. Hierarchical Storage Manager

HSM est un composant du système d'exploitation travaillant en étroite collaboration avec les méthodes d'accès ; il gère les niveaux inférieurs de la hiérarchie. Ce composant manipule les entités de type fichier et organise les transferts (migration) de ces objets vers les niveaux inférieurs (K7 ou disques optiques) en cas de faible utilisation.

Si un processus tente d'accéder un objet (fichier) migré, une opération de recall est initiée. Cette opération consiste à ramener l'objet sur son support initial. Les supports employés par ce composant sont lents (robotique, juke-box) mais disposent d'une capacité étendue (centaine de térabytes).