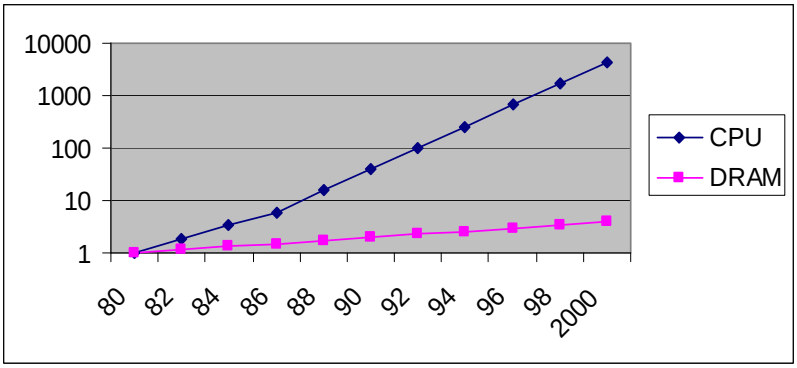
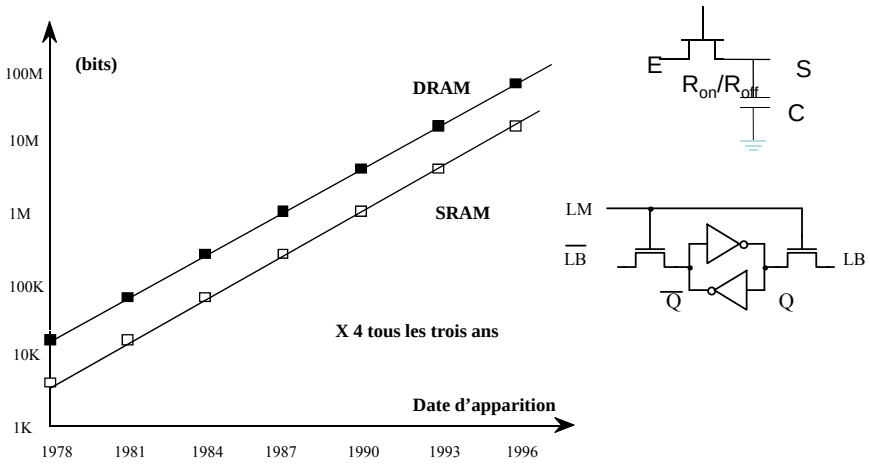
Accès aux données
Caches et alternatives

Daniel Etiemble
de@lri.fr

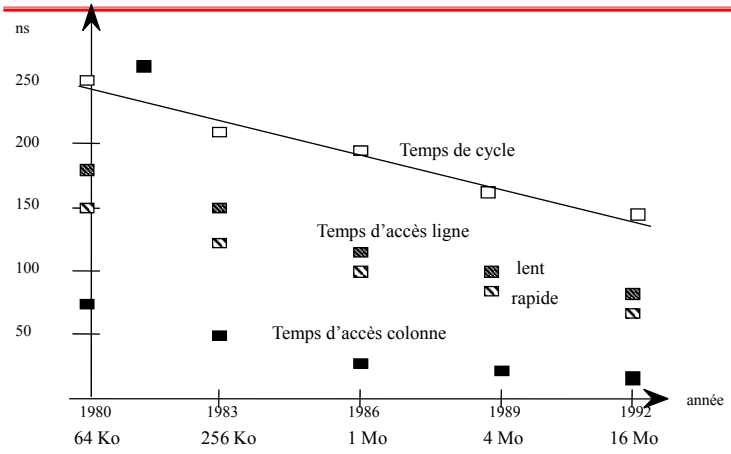
Un différentiel croissant



Capacité mémoire



Caractéristiques temporelles des DRAM

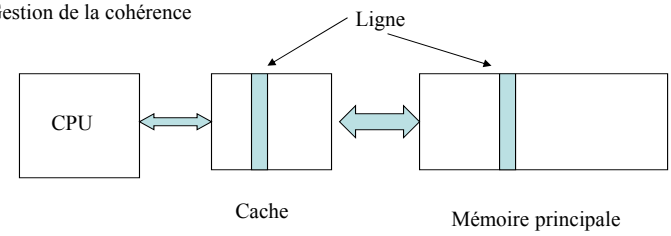


Différence SRAM-DRAM

- Taille
  - Rapport 4 en surface : 6 T / 1,5T
- Coût
  - Rapport 10 : capacité + volume de production
- Temps cycle
  - SRAM
    - Temps cycle = Temps accès
    - Evolue comme les portes de base et taille
  - DRAM
    - Lecture destructive : Temps cycle = 2 * temps d'accès
    - Temps de cycle beaucoup plus grand que SRAM

Principe des caches

- Fondés sur le principe de localité
- Mémoires de taille et vitesse différentes
- Organisation
  - Découpage en lignes (blocs)
  - Mécanisme de correspondance
    - Placement des lignes dans le cache
    - Détection succès ou échec
  - Gestion de la cohérence

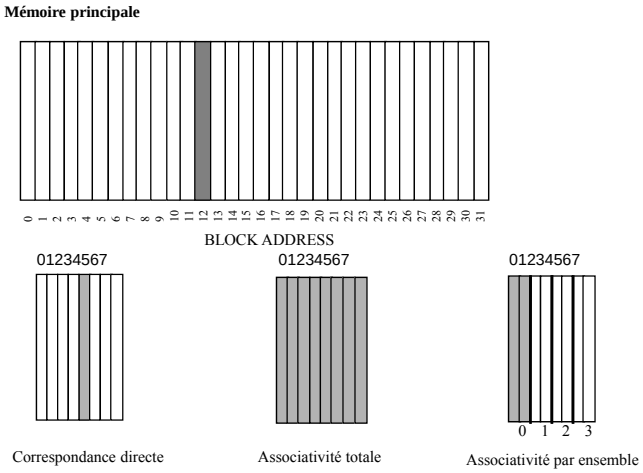


Principe de localité

- Localité spatiale
  - Si on accède à une case mémoire, on accédera à une case proche
- Localité temporelle
  - si on accède à une case mémoire, on y accédera probablement très bientôt (ou dans très longtemps !)

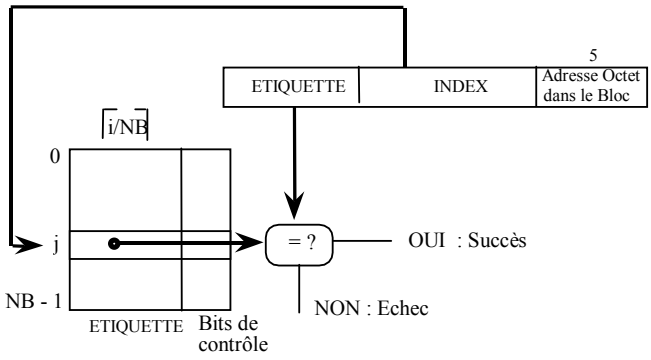
Mécanismes de correspondance

OU PLACER UNE LIGNE DE LA MP DANS LE CACHE ?



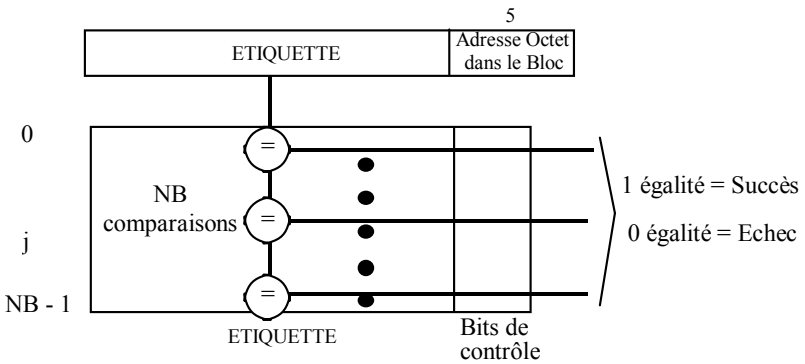
Correspondance directe

j (cache) = i (MP) mod NB, avec NB = nombre de lignes du cache



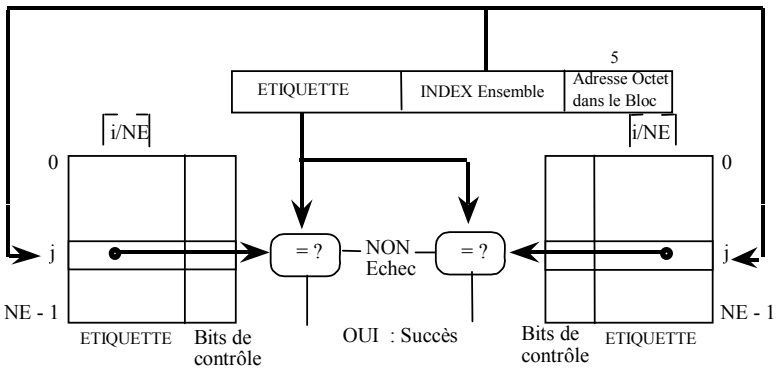
Associativité totale

Ligne i(MP) dans n'importe quelle ligne j du cache.
Etiquette = i, et NB comparateurs pour détecter succès ou échec



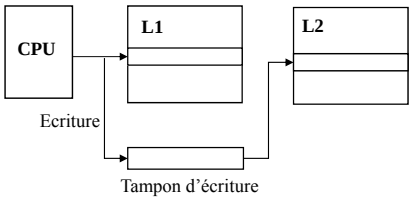
Associativité par ensemble

N (2, 4, 8) lignes par ensemble. Correspondance directe pour les ensembles et associativité à l'intérieur d'un ensemble.
N comparateurs.

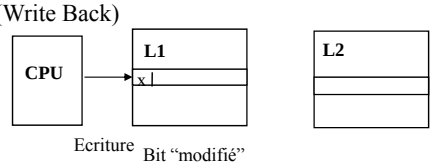


Politique en écriture

Ecriture simultanée
(Write Through)



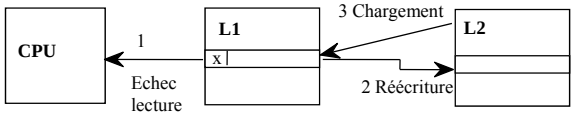
Réécriture
(Write Back)



Cohérence au plus tard

Cohérence
au plus tôt

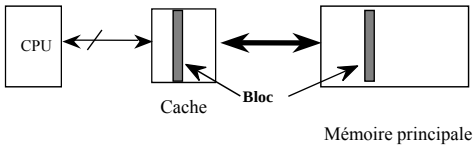
Réécriture **Réécriture quand une ligne modifiée est remplacée**



Politiques de remplacement

- Ligne à remplacer ?
  - Least Recently Used (LRU)
  - Random
  - Least Frequently Used (LFU)

CPI Mémoire



$CPI_{mémoire}$ = ma x m x p

ma: accès mémoire/instruction

m: taux d'échec

p : pénalité d'échec (cycles d'horloge)

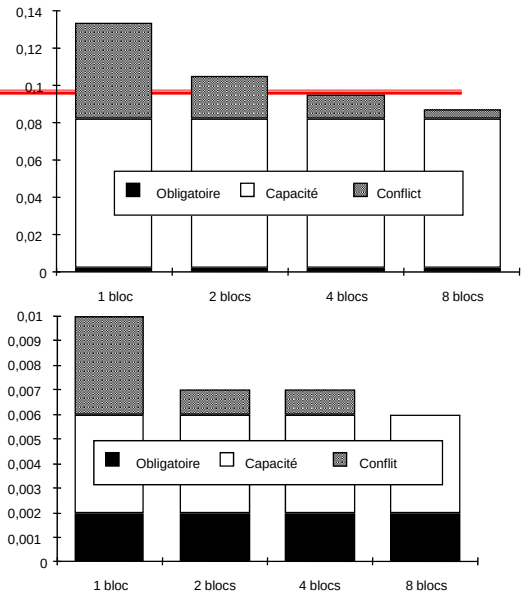
Améliorer les performances des caches

- Diminuer le taux d'échec
  - Taille (cache, ligne) et associativité
  - Optimisations du compilateur
  - Préchargement
- Diminuer le temps d'accès réussi (hit)
  - Petits caches
  - Eviter les traductions d'adresse
- Diminuer la pénalité d'échec
  - Caches non bloquants
  - Caches de second niveau

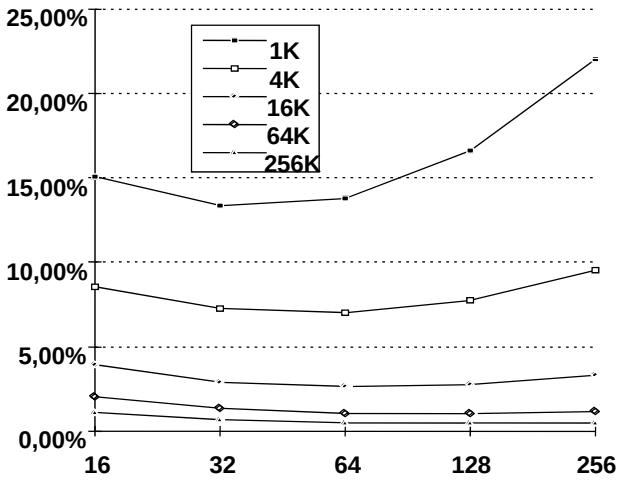
Les 3C

- Echecs de démarrage
- Echecs de capacité
- Echecs de conflit

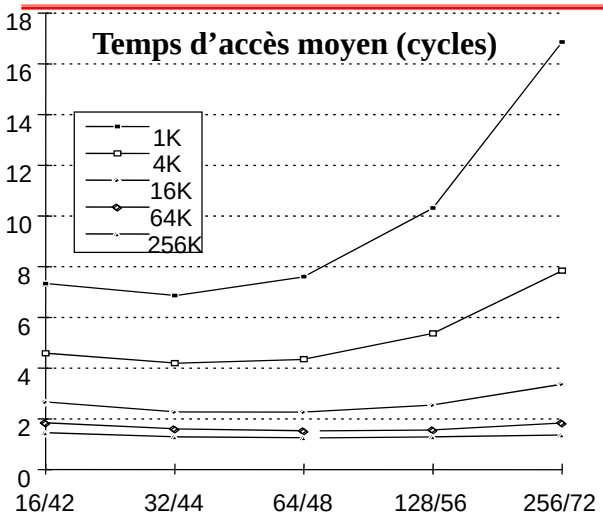
SPEC92
Lignes de 32 octets - LRU
DECStation5000



Taux d'échec (Taille de ligne)



Temps d'accès (taille de ligne)



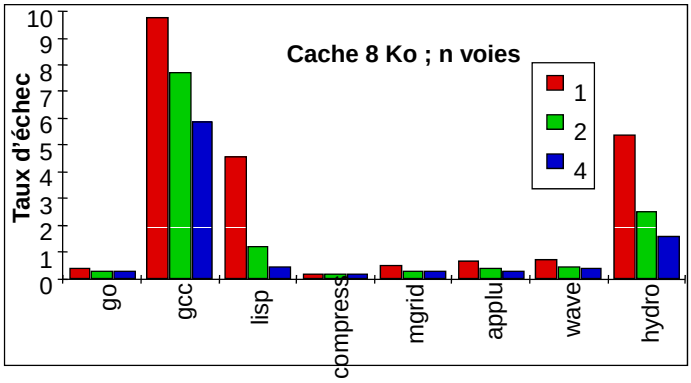
Hypothèse

40 cycles de démarrage, puis 16 octets tous les 2 cycles

Taille de ligne / Pénalité d'échec

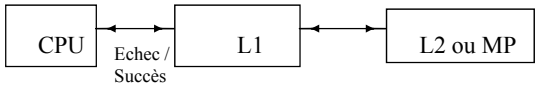
Taux d'échecs (associativité)

Le taux d'échec décroît quand l'associativité croit, mais les lectures "optimistes" ne peuvent plus être utilisées (nécessité de prédicteurs d'ensemble)

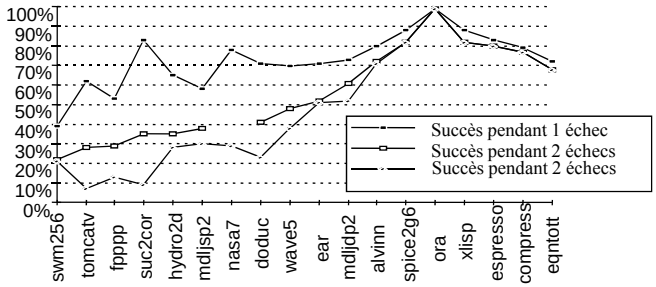


Caches non bloquants

Succès pendant échecs : le cache continue à fournir les données des accès réussis pendant qu'il traite un échec.

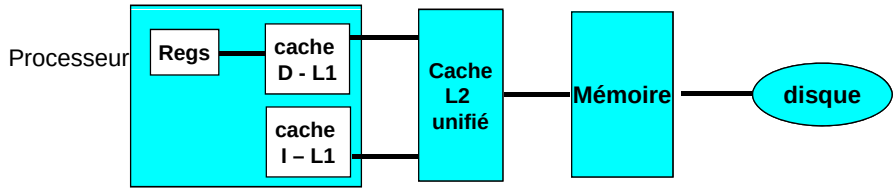


Rapport moyen entre les temps de suspension d'un cache non bloquant/cache bloquant



Plusieurs niveaux de cache

- Options: caches instructions et données séparés ou cache unifié

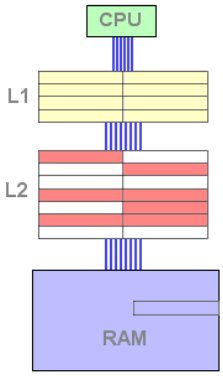


Taille :	200 o	8-64 Ko	SRAM 1-4Mo	DRAM 128 Mo	30 Go
Vitesse	<1 ns	<1 ns	6 ns	60 ns	8 ms
€/Moctet :			100 €/Mo	1.50€/Mo	0.05€/Mo
Taille ligne :	8 o	32 o	32 o	8 Ko	

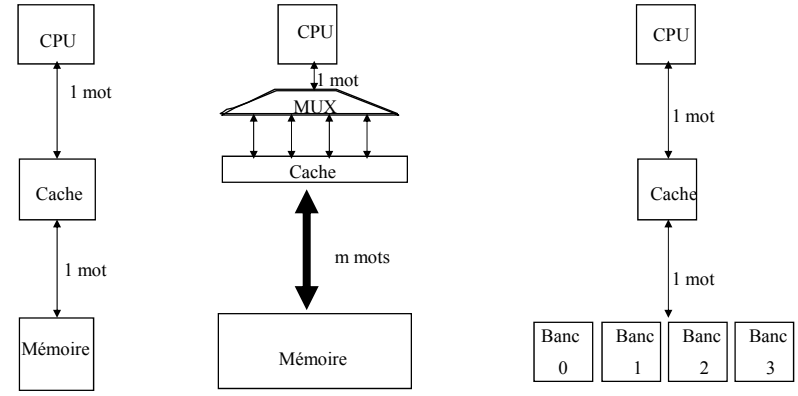
Plus gros, plus lent, moins cher

Caches inclusifs ou exclusifs

- Caches inclusifs
  - L1⊂ L2 ⊂ MP
- Caches exclusifs
  - L1 ⊄ L2
  - Fonctionnement
    - Echec L1 : bloc de MP dans L1
      - Si remplacement, bloc remplacé dans L2 et bloc demandé dans L1
    - Echec L1 et Succès L2
      - Remplacement d'un bloc dans L1 (copié dans L2) et bloc demandé de L2 dans L1
    - Nécessité d'un tampon des victimes
      - Lors d'un remplacement, le bloc remplacé est placé dans un tampon pour ne pas retarder le transfert L2 vers L1

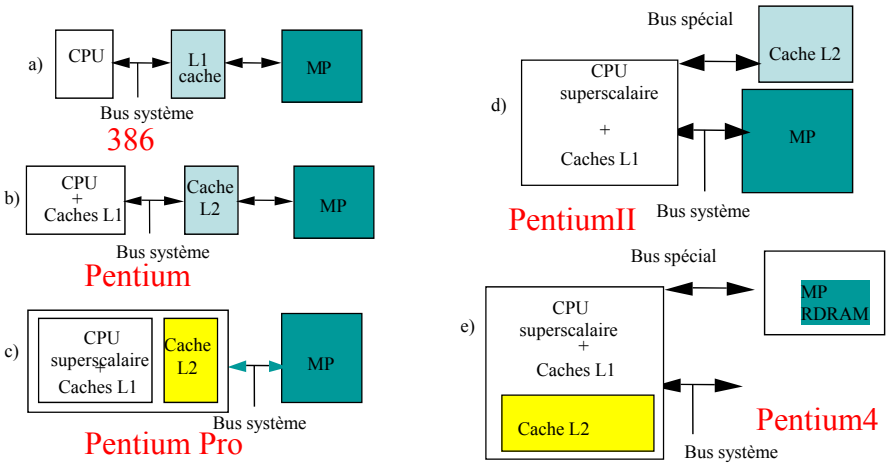


Liaison cache-mémoire principale



a) Mémoire 1 mot b) Mémoire large b) Mémoire entrelacée m bancs

Evolution des hiérarchies mémoire (1985-2000)



Caches données des processeurs Intel

CACHE PRIMAIRE

	Taille	Assoc.	Lignes	Latence	Ecriture
Pentium Pro	8 Ko	2 voies	32 oct	3	Réécriture
Pentium III	16 Ko	4 voies	32 oct	3	Réécriture
Pentium 4	8 Ko	4 voies	64 oct	2/6	Simultanée

CACHE SECONDAIRE (UNIFIE)

	Taille	Assoc.	Lignes	Latence	Ecriture
Pentium Pro	512 Ko	4 voies	32 oct		Réécriture
Pentium III*	256 Ko	8 voies	32 oct	6	Réécriture
Pentium 4	256 Ko	8 voies	128 oct	7/7	Réécriture

Amélioration des performances cache

- Réduire le taux d'échec
  - Taille de cache, de ligne et degré d'associativité ***HW***
  - Optimisations du compilateur ***SW***
  - Préchargement ***HW*** ***SW***
- Réduire le temps de l'accès réussi
  - Prédiction d'ensemble ***HW***
  - Eviter les traductions d'adresse ***HW***
- Réduire la pénalité d'échec
  - Caches non bloquants ***HW***
  - Caches de second niveau ***HW***

Code en fonction du cache

- Accès répété à des variables est favorable (localité temporelle)
- Accès avec un pas de 1 est favorable (localité spatiale)
- Exemples:
  - Cache à froid, mots de 4 octets, lignes de cache de 4 mots

```
int sumarrayrows(int a[M][N])
{
    int i, j, sum = 0;

    for (i = 0; i < M; i++)
        for (j = 0; j < N; j++)
            sum += a[i][j];
    return sum;
}
```

Taux d'échec = 1/4 = 25%

```
int sumarraycols(int a[M][N])
{
    int i, j, sum = 0;

    for (j = 0; j < N; j++)
        for (i = 0; i < M; i++)
            sum += a[i][j];
    return sum;
}
```

Taux d'échec = 100%

Exemple : multiplication de matrices

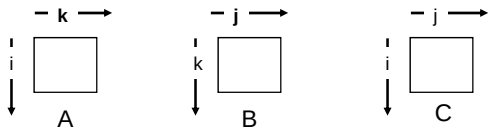
- Aspects cache à considérer
  - Taille totale du cache
    - Exploiter la localité temporelle et conserver l'ensemble de travail petit (utilisation du blocage)
  - Taille de ligne
    - Exploiter la localité spatiale
- Description:
  - Multiplication de matrice N x N
  - $O(N^3)$ opérations
  - Accès
    - N lecture par élément
    - N valeurs sommées par destination
      - Mais peuvent être conservées dans des registres

```
/* ijk */
for (i=0; i<n; i++)  {
  for (j=0; j<n; j++) {
    sum = 0.0;
    for (k=0; k<n; k++)
      sum += a[i][k] * b[k][j];
    c[i][j] = sum;
  }
}
```

sum dans un registre

Analyse du taux d'échecs pour la multiplication de matrices

- Hypothèses:
  - Taille de ligne = 32 octets (assez grand pour 4 mots de 64 bits)
  - La taille des matrices (N) est très grande
    - 1/N → 0.0
  - La taille du cache est insuffisante pour contenir plusieurs lignes
- Méthode d'analyse:
  - Examiner les accès dans la boucle interne



L'allocation mémoire des tableaux C

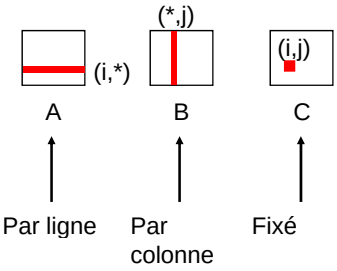
- Les tableaux C sont alloués dans l'ordre ligne d'abord
  - Ligne dans des cases mémoires contigues
- Balayage à travers les colonnes dans une ligne :
  - `for (i = 0; i < N; i++)`
    `sum += a[0][i];`
  - Accède aux éléments successifs
  - Si la taille de bloc (B) > 4 octets, exploite la localité spatiale
    - Taux d'échecs obligatoires = 4 octets / B
- Balayage à travers les lignes dans une colonne :
  - `for (i = 0; i < n; i++)`
    `sum += a[i][0];`
  - Accèdent à des éléments distants
  - Aucune localité spatiale !
    - Taux d'échecs obligatoires = 1 (i.e. 100%)

Multiplication de matrices (ijk)

```
/* ijk */
for (i=0; i<n; i++)  {
  for (j=0; j<n; j++) {
    sum = 0.0;
    for (k=0; k<n; k++)
      sum += a[i][k] * b[k][j];
    c[i][j] = sum;
  }
}
```

PRODUIT SCALAIRE

Boucle interne :



- Echecs par itération de la boucle interne:

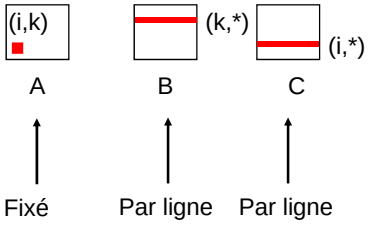
A	B	C
0,25	1,0	0,0

Multiplication de matrices (ikj)

```
/* ikj */
for (i=0; i<n; i++) {
   c[i][0]=0.0;
  for (k=0; k<n; k++) {
    r = a[i][k];
    for (j=0; j<n; j++)
      c[i][j] += r * b[k][j];
  }
}
```

SAXPY/DAXPY

Boucle interne :



- Echecs par itération de la boucle interne:

A	B	C
0,0	0,25	0,25

Multiplication ijk après transposition

```
 for (i=0; i<n; i++)
   for (j=0; j<n; j++)
        bt[j][i] = b[i][j]


/* ijk */
for (i=0; i<n; i++)  {
  for (j=0; j<n; j++) {
    sum = 0.0;
    for (k=0; k<n; k++)
      sum += a[i][k] * bt[j][k];
    c[i][j] = sum;
  }
}
```

Accès cache
~ N^2

Accès cache
~ N^3

Améliorer la localité temporelle par blocage

- Exemple : multiplication de matrices avec blocage
  - "bloc" (dans ce contexte) ne signifie pas "bloc de cache".
  - C'est un sous bloc de la matrice.
  - Exemple: N = 8; taille de sous bloc = 4

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \times \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}$$

Idée de base : Un sous bloc (i.e., $\mathbf{A_{xy}}$) peut être traité comme un scalaire.

$$C_{11} = A_{11}B_{11} + A_{12}B_{21} \qquad C_{12} = A_{11}B_{12} + A_{12}B_{22}$$

$$C_{21} = A_{21}B_{11} + A_{22}B_{21} \qquad C_{22} = A_{21}B_{12} + A_{22}B_{22}$$

Multiplication par bloc (bijk)

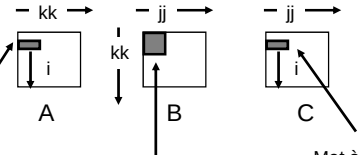
```
for (jj=0; jj<n; jj+=bsize) {
  for (i=0; i<n; i++)
    for (j=jj; j < min(jj+bsize,n); j++)
      c[i][j] = 0.0;
  for (kk=0; kk<n; kk+=bsize) {
    for (i=0; i<n; i++) {
      for (j=jj; j < min(jj+bsize,n); j++) {
        sum = 0.0
        for (k=kk; k < min(kk+bsize,n); k++) {
          sum += a[i][k] * b[k][j];
        }
        c[i][j] += sum;
      }
    }
  }
}
```


Analyse de la multiplication de matrices par blocs

- La boucle interne multiplie une tranche de *A* (*1 X tailleb)* par un bloc de *B* *(taille_b X taille_b)* et accumule dans une tranche de C (*1 X taille_b*)
- *i* étapes à travers *n* tranches de lignes of *A* & *C*, en utilisant le même *B*

```
for (i=0; i<n; i++) {
  for (j=jj; j < min(jj+bsize,n); j++) {
    sum = 0.0
    for (k=kk; k < min(kk+bsize,n); k++) {
      sum += a[i][k] * b[k][j];
    }
    c[i][j] += sum;
  }
```

Boucle interne



Tranche de ligne accédée *Taille_b* fois

Bloc réutilisé n fois

Met à jour les éléments successifs de la tranche

Optimisations du compilateur

Améliore la localité spatiale

ECHANGE DE BOUCLES

```
for (j=0; j<100; j++)
    for (i=0; i<5000; i++)
        x[i][j] = 2*x[i][j];
```

```
for (i=0; i<5000; i++)
    for (j=0; j<100; j++)
        x[i][j] = 2*x[i][j];
```

FUSION DE TABLEAUX

```
int val[SIZE];
int key[SIZE];
```

```
struc merge {
        int val;
;       int key; }
struc merge merge-array [SIZE];
```

Optimisations du compilateur

Améliore la localité temporelle

FUSION DE BOUCLES

```
for (i=0; i<N; i++)
    for (j=0; j<N; j++)
        a[i][j] = 1/b[i][j]*c[i][j];

for (i=0; i<N; i++)
    for (j=0; j<N; j++)
        d[i][j] = a[i][j]+c[i][j];
```

```
for (i=0; i<N; i++)
    for (j=0; j<N; j++)
        {
        a[i][j] = 1/b[i][j]*c[i][j];
        d[i][j] = a[i][j]+c[i][j];
        }
```

Optimisations du compilateur

BLOCAGE

AMELIORE LA LOCALITE TEMPORELLE

```
for (i=0; i<N; i++)
    for (j=0; j<N; j++)
        {r=0;
        for (k=0; k<N; k++)
            r+=y[i][k]*z[k][j];
        x[i][j]=r;}
```

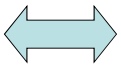
```
for (jj=0; jj<N; jj+=B)
for (kk=0; kk<N; kk+=B)
for (i=0; i<N; i++)
    for (j=jj; j<(min(jj+B-1,N); j++)
        {r=0;
        for (k=kk; j<(min(kk+B-1,N); k++)
            r+=y[i][k]*z[k][j];
        x[i][j]+=r;}
```

B : facteur de blocage

Compatibilités entre optimisations

- Pas unitaire est efficace pour les caches et pour la vectorisation
- Mais contre-exemples

```
for (j=0; j<2; j++)
    for (i=0; i<1000; i++)
        x[i][j] = 0.0;
```



```
for (i=0; i<1000; i++)
    for (j=0; j<2; j++)
        x[i][j] = 0.0;
```

Le tableau tient dans le cache
Réutilisation totale

Boucle interne trop courte

```
for (j=1; j<m; j++)
    for (i=0; i<n; i++)
        x[i][j] = x[i][j] + x[i][j-1];
```

```
for (i=0; i<n; i++ )
    for (j=1; j<m; j++)
        x[i][j] = x[i][j] + x[i][j-1];
```

Bon comportement du cache, mais récursion sur j

Préchargement matériel

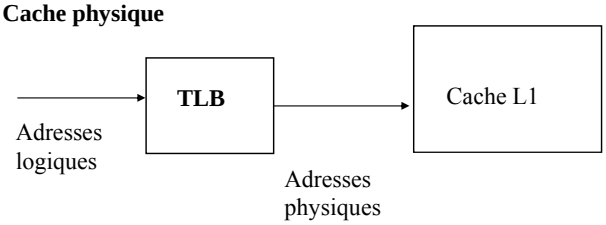
- Précharger les instructions ou les données avant qu'elles ne soient nécessaires (charger une ligne avant un échec cache)
- Quelle ligne ?
  - La suivante (préchargement séquentiel selon la localité spatiale)
  - Une ligne prédite
- Quand ?
  - Toujours
  - Sur un échec
  - Sur un échec et lorsqu'on accède une donnée préchargée
- Où ?
  - Dans le cache (pollution potentielle)
  - Dans un tampon

Préchargement logiciel

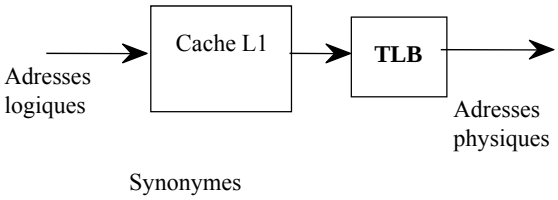
- Instructions de préchargement (IA-32)
  - PREFETCHT0 m8 : tous les niveaux de cache
  - PREFETCHT1 m8: tous les niveaux sauf L0
  - PREFETCHT2 m8: tous les niveaux sauf L0 et L1
  - PREFETCHNTA m8: préchargement dans une structure non temporelle
- Indications de préchargement
  - Dépendent des implémentations

Cache et TLB

Cache physique

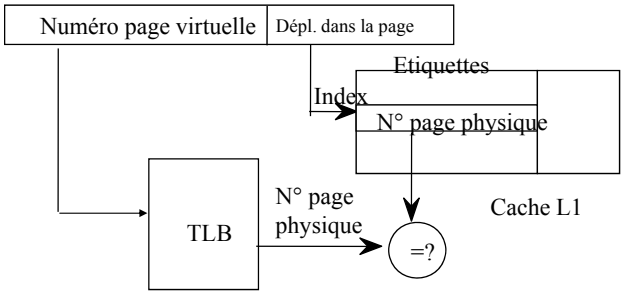


Cache virtuel



Cache et TLB

Caches avec index virtuels et adresses physiques



TLB et cache sont accédés en parallèle
Avec la correspondance directe, taille du cache = taille de page