Módulo 4. Jerarquía de memoria: Cache

Grado en Ingeniería Informática

Curso 2015-2016

Índice

- Introducción
- Jerarquía de memoria
- Principio de localidad
- Memorias cache
- Políticas de emplazamiento
- Políticas de actualización
- Políticas de reemplazamiento
- Rendimiento
- ¿Cómo mejorar el rendimiento de la cache?
  - Reducir la tasa de fallos
  - Reducir la penalización del fallo
  - Reducir el tiempo de acierto (hit time)
  - Aumentar el ancho banda

Índice

- **Bibliografía**

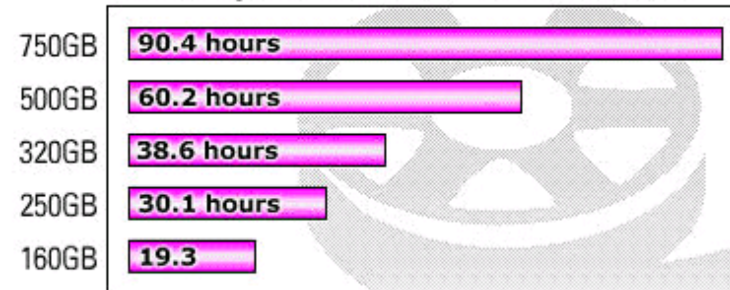
  – John L. Hennessy & David A. Patterson , “Computer Architecture: A Quantitative Approach”, Morgan Kaufmann, 4ª o 5ª ed.

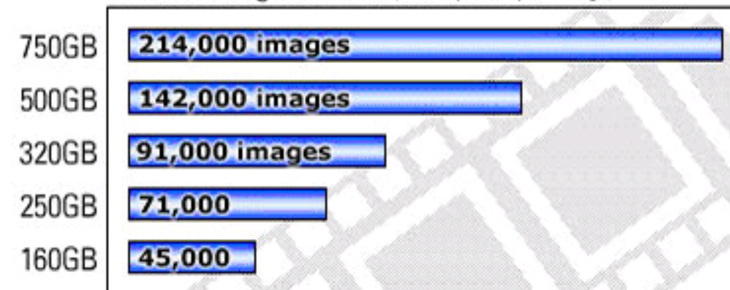
  – David A. Patterson & John L. Hennessy, “Computer Organization and Design. The Hardware/Software Interface”, Morgan Kaufmann 5ª ed.

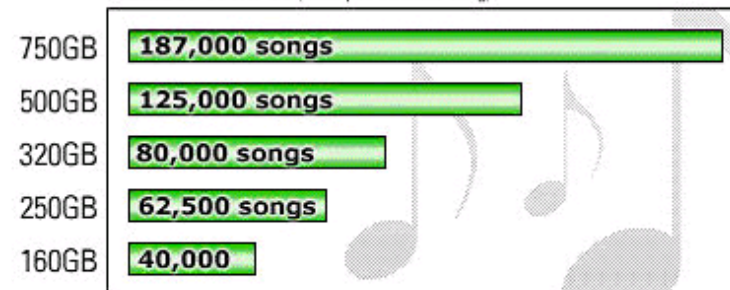
  – B. Jacob, S.W. Ng, D.T. Wong, “Memory Systems. Cache, DRAM, Disk”, 1ª ed. Morgan Kaufmann.

Introducción

- Los datos ocupan espacio en memoria
- Acceder a ellos implica un cierto tiempo







Introducción

- Valores típicos de tiempos de acceso y precios por Gbyte (año 2012)

Tipo de memoria	Tiempo de acceso (ns)	$ por Gbyte (2008)	$ por Gbyte (2012)
SRAM	0.5-2.5	2000-5000	500-1000
DRAM	50-70	20-75	10-20
Flash	5.000-50.000		0,75-1,00
Disco magnético	5.000.000-20.000.000	0.20-2	0,05-0,10

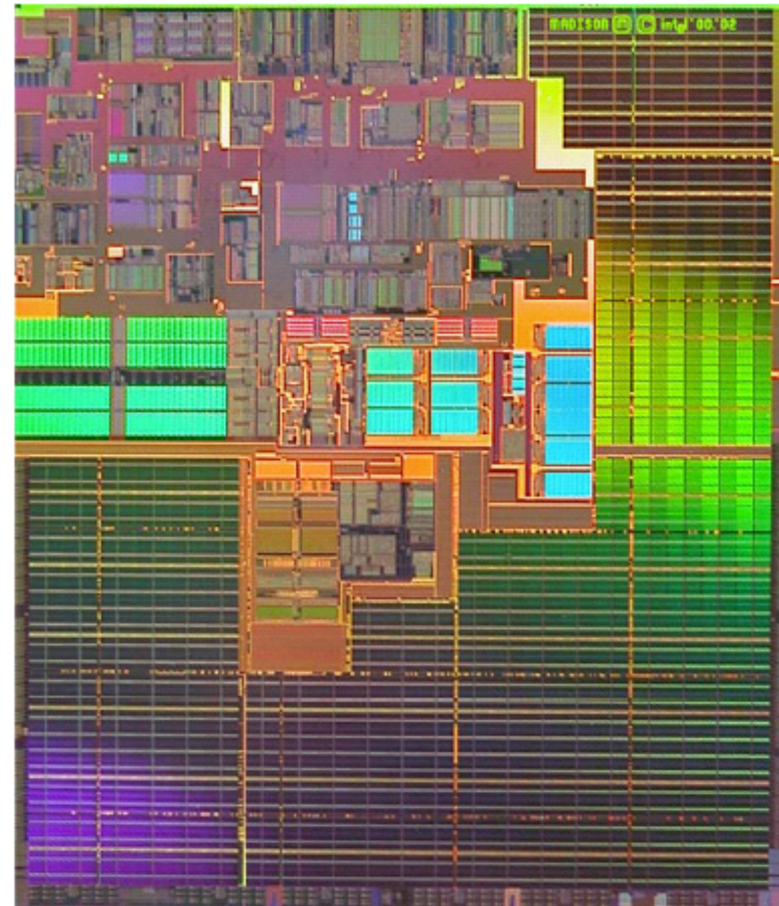
Patterson & Hennessy 5º ed

Introducción

Tamaño de la cache
Del 50 al 75 % del área. Más del 80% de los transistores

PentiumIII

Latencia:
1ciclo (Itanium2) a 3 ciclos Power4-5

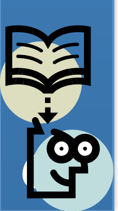


Itaniun 2 Madison

ec

Jerarquía de Memoria

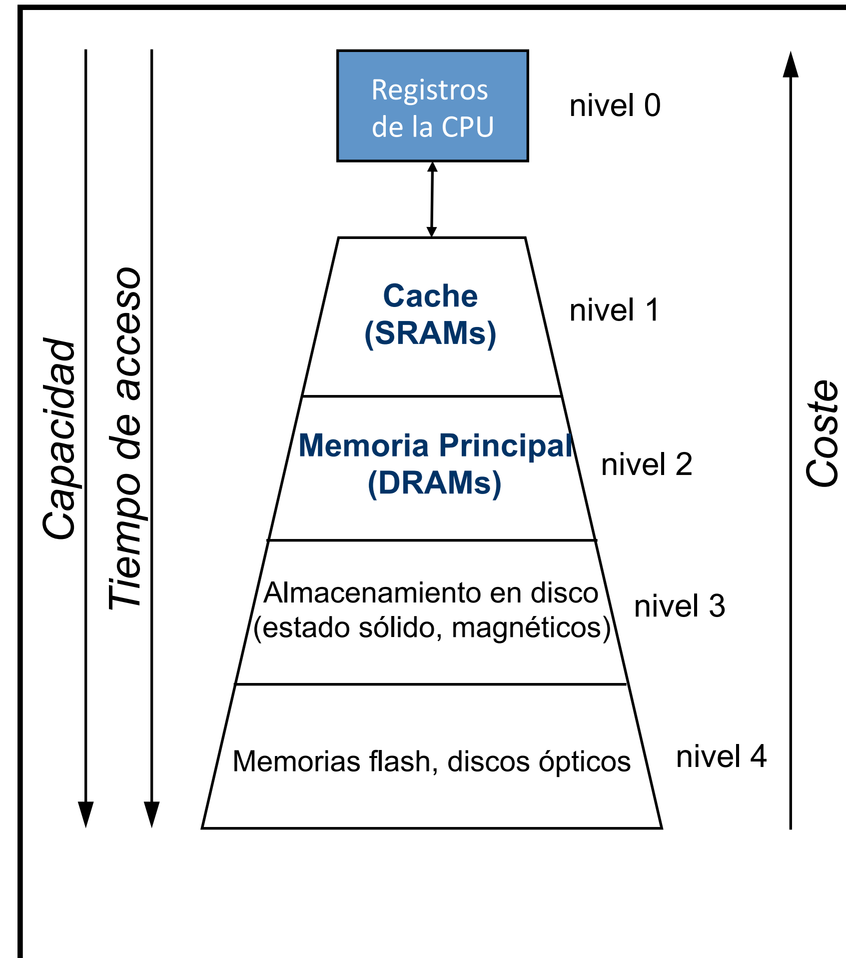
- Objetivo:
  - Conseguir una memoria de gran tamaño, rápida y al menor coste posible.
  - De forma transparente al usuario
- ¿Cómo organizar la memoria?
- Base:
  - Principio de localidad
    - "Cualquier programa accede a una porción relativamente pequeña de su espacio de direcciones en cualquier instante de tiempo"

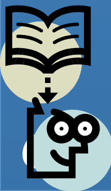


Jerarquía de memoria

Niveles de la Jerarquía de memoria

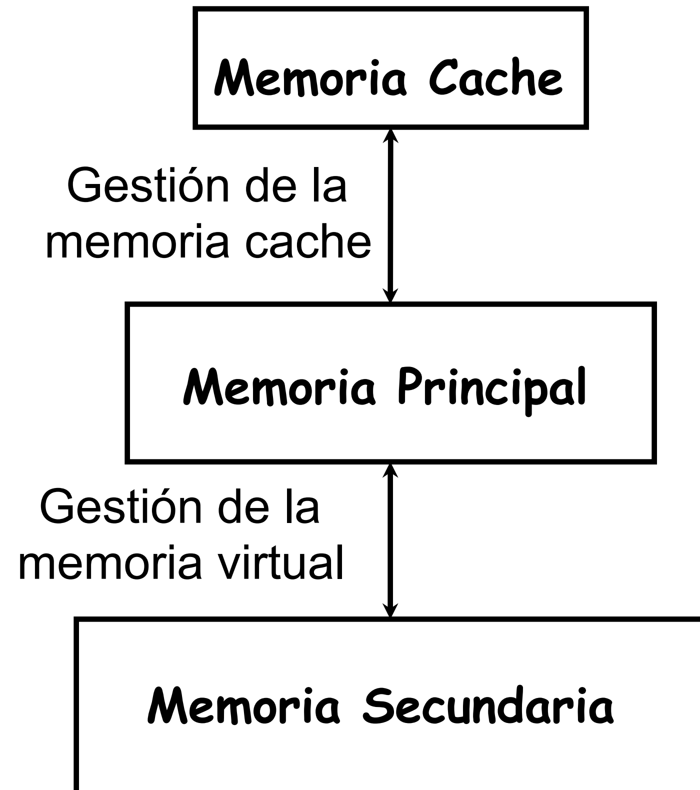
- Un computador típico está formado por diversos niveles de memoria, *organizados de forma jerárquica*:
  - Registros de la CPU
  - Memoria Cache
  - Memoria Principal
  - Memoria Secundaria (discos)
  - Memorias flash y CD-ROMs
- El coste de todo el sistema de memoria excede al coste de la CPU
  - Es muy importante optimizar su uso





Jerarquía de memoria

- **Gestión de la memoria cache**
  - Controla la transferencia de información entre la memoria cache y la memoria principal
  - Suele llevarse a cabo mediante hardware específico (MMU o "Memory Management Unit")
- **Gestión de la memoria virtual**
  - Controla la transferencia de información entre la memoria principal y la memoria secundaria
  - Parte de esta gestión se realiza mediante hardware específico (MMU) y otra parte la realiza el S.O



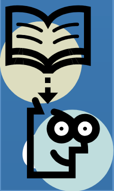
Jerarquía de memoria

- Propiedades de la jerarquía de memoria:
  - Inclusión
    - Cualquier información almacenada en el nivel de memoria Mi, debe encontrarse también en los niveles Mi+1, Mi+2, …, Mn.

      es decir: $M1 \subset M2 \subset \ldots \subset Mn$
  - Coherencia
    - Las copias de la misma información existentes en los distintos niveles deben ser coherentes
    - Si un bloque de información se modifica en el nivel Mi, deben actualizarse los niveles Mi+1,.., Mn
  - Localidad
    - Las referencias a memoria generadas por la CPU, para acceso a datos o a instrucciones, están concentradas o agrupadas en ciertas regiones del tiempo y del espacio

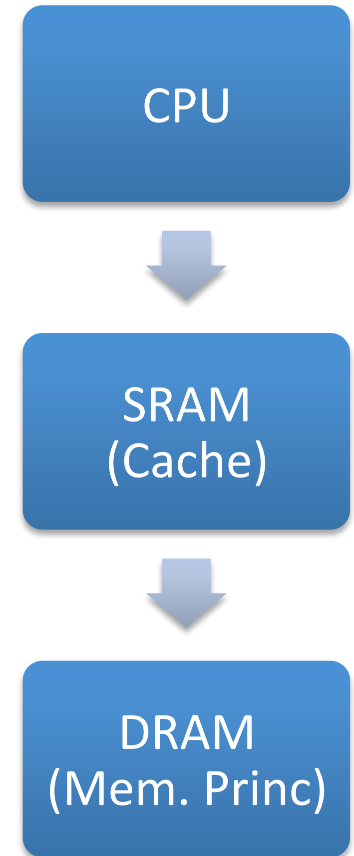
Principio de localidad

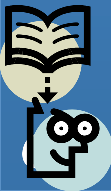
- Localidad temporal: Un dato usado en un determinado instante tiende a ser reutilizado pronto
  - Es habitual que un programa acceda a la misma variable varias veces. Lo óptimo es mantenerla en registro, pero no siempre es posible
- Localidad espacial: Si un dato es utilizado en un determinado instante, es muy probable que los datos cercanos a él sean también pronto utilizados
  - Es habitual que un programa acceda a datos que están almacenados en posiciones consecutivas
  - Los arrays se almacenan en posiciones consecutivas y se suelen acceder secuencialmente



¿Cómo explotar la localidad?

- Cada vez que se accede a memoria principal, llevamos una copia del dato a una memoria más pequeña y rápida (SRAM)
  - Introduce una penalización en el primer acceso
- Será amortizada si se producen más accesos, pues se usará la copia y se evitará un acceso a memoria principal.
- Pero además, se copian varios datos más
  - Por ejemplo, si se accede al elemento a[0], se traen a la cache a[0],a[1],a[2] y a[3]
  - De nuevo, hay una penalización inicial que suele amortizarse.





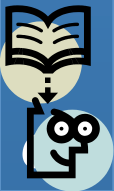
Gestión de la jerarquía de memoria

- Vamos a trabajar con una memoria en 2 niveles
  - Cache y memoria principal

- **Bloque**: unidad mínima de transferencia entre los dos niveles
- **Acierto (hit)**: el dato solicitado está en la cache
  - ***Tasa de aciertos*** (hit ratio): la fracción de accesos encontrados en el nivel i
  - ***Tiempo de acierto*** (hit time): tiempo de detección de acierto + tiempo de acceso del nivel i. (Tiempo total invertido para obtener un dato cuando éste se encuentra en el nivel i)
- **Fallo (miss)**: el dato solicitado no está en la cache y es necesario buscarlo en la memoria principal
  - ***Tasa de fallos*** (miss ratio): 1 - (Tasa de aciertos)
  - ***Tiempo de penalización por fallo*** (miss penalty): tiempo invertido para mover un bloque del nivel i+1 al nivel i, cuando el bloque referenciado no está en el nivel i.

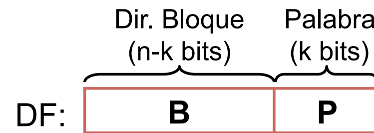
Gestión de la jerarquía de memoria

- Cuando la CPU genera una referencia, busca en la cache
  - Si la referencia no se encuentra en la cache: **FALLO**
  - Cuando se produce un fallo, no solo se transfiere una palabra, sino que se lleva un **BLOQUE** completo de información de la MP a la MC
  - *Por la propiedad de localidad temporal*
    - Es probable que en una próxima referencia se direccione la misma posición de memoria
    - Esta segunda referencia no producirá fallo: producirá un **ACIERTO**
  - *Por la propiedad de localidad espacial*
    - Es probable que próximas referencias sean direcciones que pertenecen al mismo bloque
    - Estas referencias no producen fallo

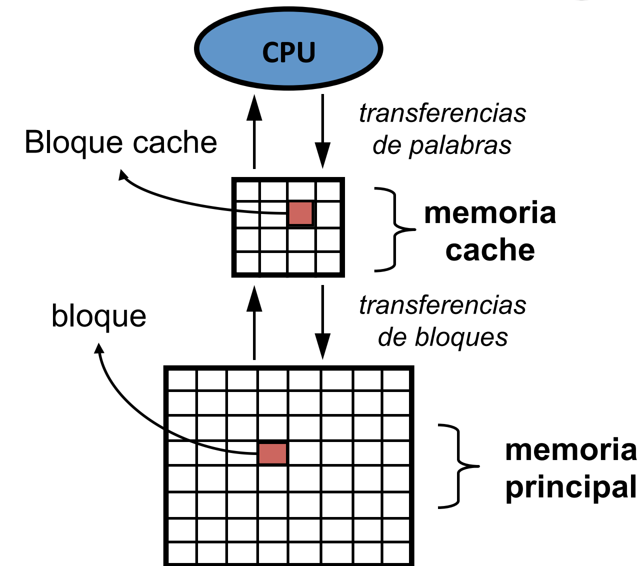


Memoria Cache

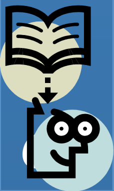
- **Estructura** del sistema memoria cache/principal:
  - *MP (memoria principal):*
    - formada por 2^n palabras direccionables "dividida" en nB bloques de tamaño fijo de 2^K palabras por bloque
    - Campos de una dirección física:



  - *MC (memoria cache):*
    - formada por nM bloques (o líneas) de 2^K palabras cada uno (nM<<nB)
  - *Directorio (en memoria cache):*
    - Para cada bloque de MC, indica cuál es el bloque de MP que está alojado en él.



nB: número de bloques
nM: número de bloques de cache
B: dirección del bloque
M: dirección del bloque de cache
P: palabra dentro del bloque



Memoria cache

Dirección de memoria

CPU

Memoria cache

Buscar en el directorio

¿Está en cache el bloque?

NO (fallo)

Obtener el bloque de MP

Asignar bloque cache al bloque de MP

Almacenar bloque en bloque cache

SI (acierto)

Acceder a bloque cache

Seleccionar palabra

FIN

tiempo de identificación

tiempo de penalización

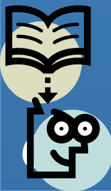
tiempo de acierto

tiempo de acceso

- Maximizar la tasa de aciertos
- Minimizar el tiempo de acceso
- Minimizar el tiempo de penalización
- Reducir el coste hardware

$$T_{total} = T_{acierto} + (1 - H) T_{penalizacion}$$

(frecuencia de hits)

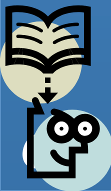


Memoria Cache

- ¿Cómo sabemos que un dato está en la cache?
- Y si está, ¿cómo lo encontramos?

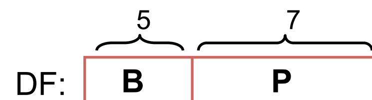
POLÍTICAS DE EMPLAZAMIENTO

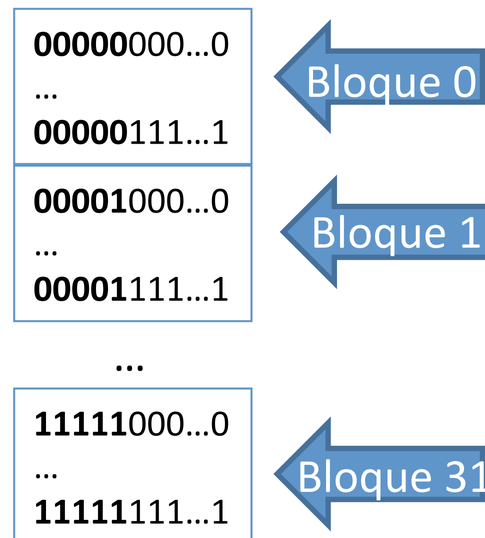
- Política de emplazamiento:
  - Necesaria ya que existen menos bloques en MC que bloques en MP
  - Determina en qué bloque, o bloques, de MC, puede cargarse cada bloque de MP
- Existen diferentes políticas:
  - Emplazamiento directo
  - Emplazamiento asociativo
  - Emplazamiento asociativo por conjuntos

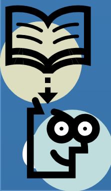


Políticas de emplazamiento

- Para todos los ejemplos:
  - Tamaño de bloque 128 palabras => k=7
  - Memoria cache con 8 bloques
  - Memoria principal 4k palabras => n = 12







Emplazamiento directo

- Cada bloque B de memoria principal tiene asignado un **único** bloque cache M en donde ubicarse.
- Este bloque cache viene dado por la expresión: **M = B mod nM**.
- Si $nM = 2^m$ entonces M = (m bits menos significativos de B)

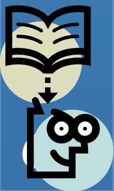
Nº Bloque Cache

000	001	010	011	100	101	110	111

Cache

Memoria

Nº Bloque: 00001, 00101, 01001, 01101, 10001, 10101 ....



Emplazamiento directo

Dirección ofrecida por la CPU

Posible dirección cache

B	P

ETIQUETA	M	P

k

n-m-k

m

V	ETIQUETA

DATOS

V	ETIQUETA

DATOS

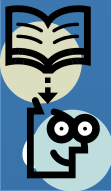
V	ETIQUETA

=

Bit de validez

acierto

- El acceso al bloque cache y al directorio es directo.
- Para conocer si un bloque de memoria principal está cargado en MC, basta con comparar las etiquetas

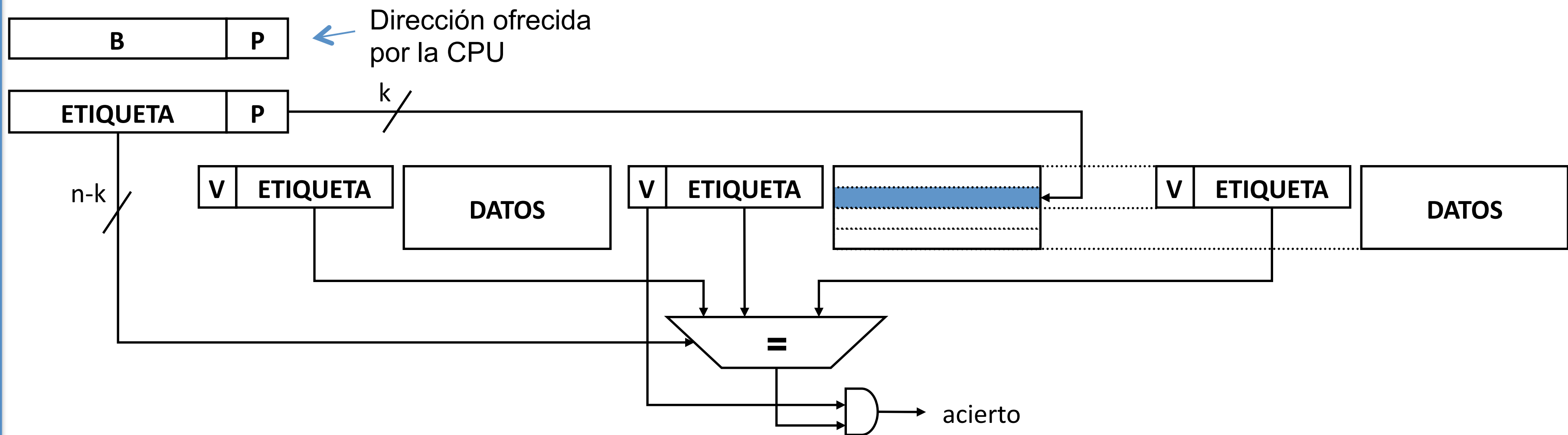


¿Cuánto bits tiene m para los datos del ejemplo?

Emplazamiento asociativo

- Cada bloque B puede ubicarse en cualquiera de los nM bloques de cache

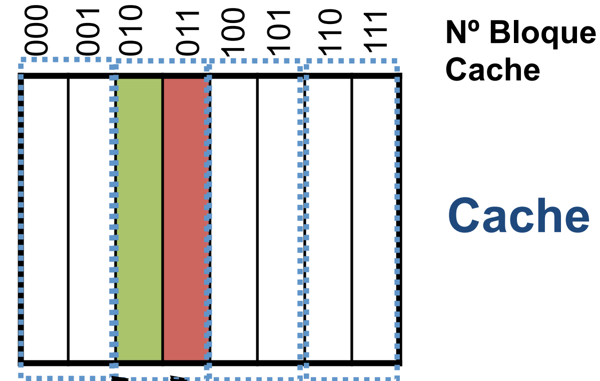
Emplazamiento asociativo

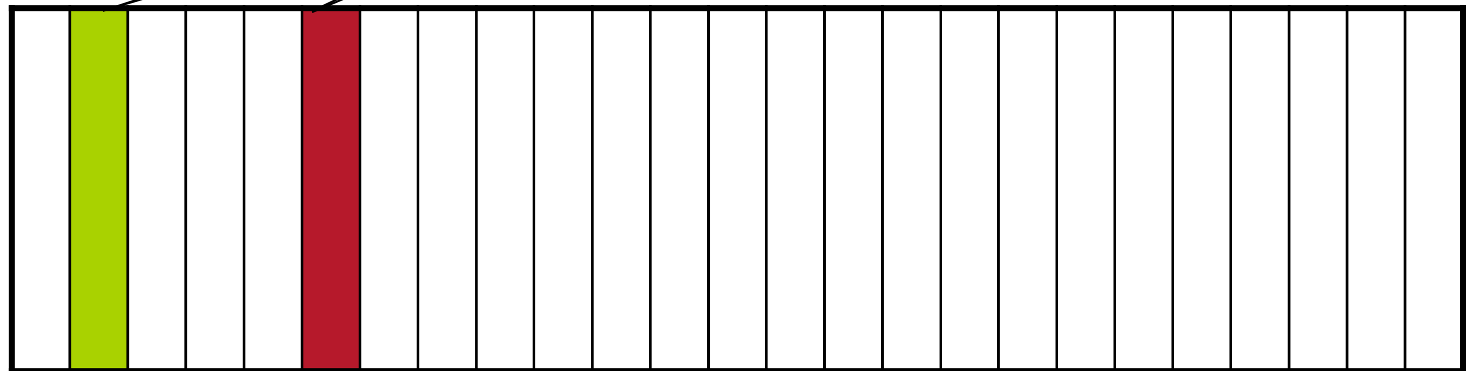


- Para conocer si un bloque de memoria principal está cargado en MC, hay que comparar todas las etiquetas

Emplazamiento asociativo por conjuntos

- Cada bloque B tiene asignado un conjunto fijo **C** y puede ubicarse en cualquiera de los marcos que componen dicho conjunto
- Este conjunto viene dado por la expresión: **C = B mod nC**





Emplazamiento asociativo por conjuntos

B | P

Dirección ofrecida por la CPU

ETIQUETA | C | P

k

n-m-k

m

V | ETIQUETA | DATOS

. . .

C_0

C_1

C_{n-1}

=

acierto

- El directorio almacena para cada marco una etiqueta con los n-k-m bits que completan la dirección del bloque almacenado
- El acceso al conjunto es directo y al marco dentro del conjunto asociativo

¿Cuánto vale m para los datos del ejemplo?

Emplazamiento asociativo por conjuntos

- El valor nM/nC se denomina grado de asociatividad o número de vías de la MC
  - Grado de asociatividad = 1, equivale a emplazamiento directo
  - Grado de asociatividad = nM, equivale a emplazamiento asociativo

Políticas de emplazamiento

Emplazamiento	Ventajas	Inconvenientes
Directo	Acceso simple y rápido	Alta tasa de fallos cuando varios bloques compiten por el mismo marco
Asociativo	Máximo aprovechamiento de la MC	Una alta asociatividad impacta directamente en el acceso a la MC
Asociativo por conjuntos	Es un enfoque intermedio entre emplazamiento directo y asociativo. El grado de asociatividad afecta al rendimiento, al aumentar el grado de asociatividad disminuyen los fallos por competencia por un marco Grado óptimo: entre 2 y 16 Grado más común: 2	Al aumentar el grado de asociatividad aumenta el tiempo de acceso y el coste hardware

Política de actualización

- **¿Qué ocurre cuando se produce una escritura en memoria?**
  - ¿Se escribe sólo en la cache? ¿Sólo en la memoria principal? ¿En las dos?

- **Write-through:** Todas las escrituras actualizan la cache y la memoria
  - Al reemplazar, se puede eliminar la copia de cache: Los datos están ya en la memoria
  - Bit de control en la cache: Sólo un bit de validez

- **Write-back:** Todas las escrituras actualizan sólo la cache
  - Al reemplazar, no se pueden eliminar los datos de la cache: Deben ser escritos primero en la memoria de siguiente nivel
  - Bit de control: Bit de validez y bit de sucio

Política de actualización

- **¿Qué ocurre cuando se produce una fallo de escritura?**

- **Write allocate** (con asignación en escritura): en un fallo de escritura se lleva el bloque que se va a escribir a la cache

- **No-write allocate** (sin asignación en escritura): en un fallo de escritura el dato sólo se modifica en la MP (o nivel de memoria siguiente)

Políticas de actualización

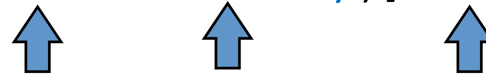
- Comparación:
  - Write-through:
    - La memoria siempre tiene el último valor
    - Control simple
  - Write-back:
    - La memoria puede no contener el valor actualizado
    - Mucho menor ancho de banda necesario, escrituras múltiples en bloque
    - Mejor tolerancia a la alta latencia de la memoria

Políticas de reemplazamiento

- **¿Qué ocurre si la MC está llena? ¿Qué bloque de cache se debe desalojar?**
- Espacio de reemplazamiento: conjunto de posibles bloques que pueden ser reemplazados por el nuevo bloque
  - **Directo**: el bloque que reside en el marco que el nuevo bloque tiene asignado. Al no existir alternativas no se requieren algoritmos de reemplazamiento
  - **Asociativo**: cualquier bloque que resida en la cache
  - **Asociativo por conjuntos**: cualquier bloque que resida en el conjunto que el nuevo bloque tiene asignado
- Algoritmos (implementados en hardware):
  - **Aleatorio**: se escoge un bloque del espacio de reemplazamiento al azar
  - **FIFO**: se sustituye el bloque del espacio de reemplazamiento que lleve más tiempo cargado
  - **LRU** (least recently used): se sustituye el bloque del espacio de reemplazamiento que lleve más tiempo sin haber sido referenciado
  - **LFU** (least frequently used): se sustituye el bloque del espacio de reemplazamiento que haya sido referenciado en menos ocasiones

Rendimiento

- Procesador con memoria perfecta (ideal)
  - Tcpu = N x CPI x Tc
  - Como N x CPI = Nº ciclos CPU --> Tcpu = Nº ciclos CPU x Tc
- Procesador con memoria real
  - Tcpu = (Nº ciclos CPU + Nº ciclos espera memoria) x Tc
  - Cuántos ciclos de espera por la memoria?
    - Nº ciclos espera memoria = Nº fallos x Miss Penalty
    - Nº fallos = Nº ref a memoria x Miss Rate
    - Nº ref a memoria = N x AMPI(Media de accesos a memoria por instrucción)
  - Luego:
    - Nº ciclos espera memoria = N x AMPI x Miss Rate x Miss Penalty
  - Y finalmente
    - Tcpu = [(N x CPI) + (N x AMPI x Miss Rate x Miss Penalty)] x Tc
    - Tcpu = N x [CPI + (AMPI x Miss Rate x Miss Penalty)] x Tc



Define el espacio de diseño para la optimización de Mc

Tc = tiempo de ciclo, N = nº de instrucciones

Rendimiento

- Penalización media por instrucción
  - Comparando
    - Tcpu = N x CPI x tc
    - Tcpu = N x [CPI + (AMPI x Miss Rate x Miss Penalty)] x Tc

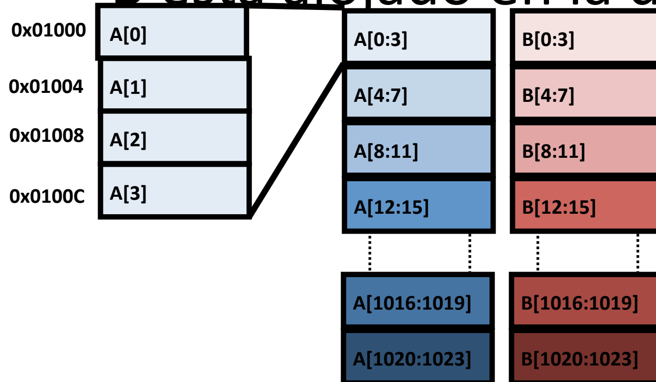
    se pueden definir los ciclos de penalización media por instrucción debida al comportamiento de la memoria:
    - Penalización media por instrucción = AMPI x Miss Rate x Miss Penalty

- Tiempo medio de acceso a memoria (TMAM)
    - TMAM = Tiempo invertido en accesos a memoria / Nº accesos =
      = (T de accesos a Mc + T de penalización por fallos) / Nº accesos =
      = Hit time + (Nº de fallos x Miss Penalty / Nº accesos)
    - TMAM = Hit time + Miss Rate x Miss Penalty

Rendimiento

- Ejemplo:
  - Se tienen dos arrays de 1024 palabras de 32 bits, se accede a los arrays para ejecutar el siguiente código:

    valor = A[i]+B[i]
  - Se tiene una memoria cache de 128 Bytes con bloques de 16 Bytes
  - A está alojado en la dirección: 0x01000
  - B está alojado en la dirección: 0x04000



a) Acceso directo

b) Asociativo con 2 vías (LRU)

c) Asociativo con 4 vías (FIFO)

d) Completamente asociativa

Rendimiento

- ¿Por qué se producen los fallos de cache? Las 3 C's
  - Iniciales (Compulsory)
    - Causados por la primera referencia a un bloque que no está en la cache → Hay que llevar primero el bloque a la cache
    - Inevitables, incluso con cache infinita
    - No depende del tamaño de la Mc. Sí del tamaño de bloque.
  - Capacidad
    - Si la cache no puede contener todos los bloques necesarios durante la ejecución de un programa, habrá fallos que se producen al recuperar de nuevo un bloque previamente descartado
  - Conflicto
    - Un bloque puede ser descartado y recuperado de nuevo porque hay otro bloque que compite por la misma línea de cache (aunque haya otras líneas libres en la cache)
    - No se pueden producir en caches puramente asociativas.

Políticas de emplazamiento

Ejemplo: Valores promedio de Miss Rate para programas de SPEC92

- Reemplazamiento: LRU
- Tamaño bloque: 32 bytes (constante)
- Variando: tamaño cache, asociatividad

Miss rate

0.14
0.12
0.1
0.08
0.06
0.04
0.02
0

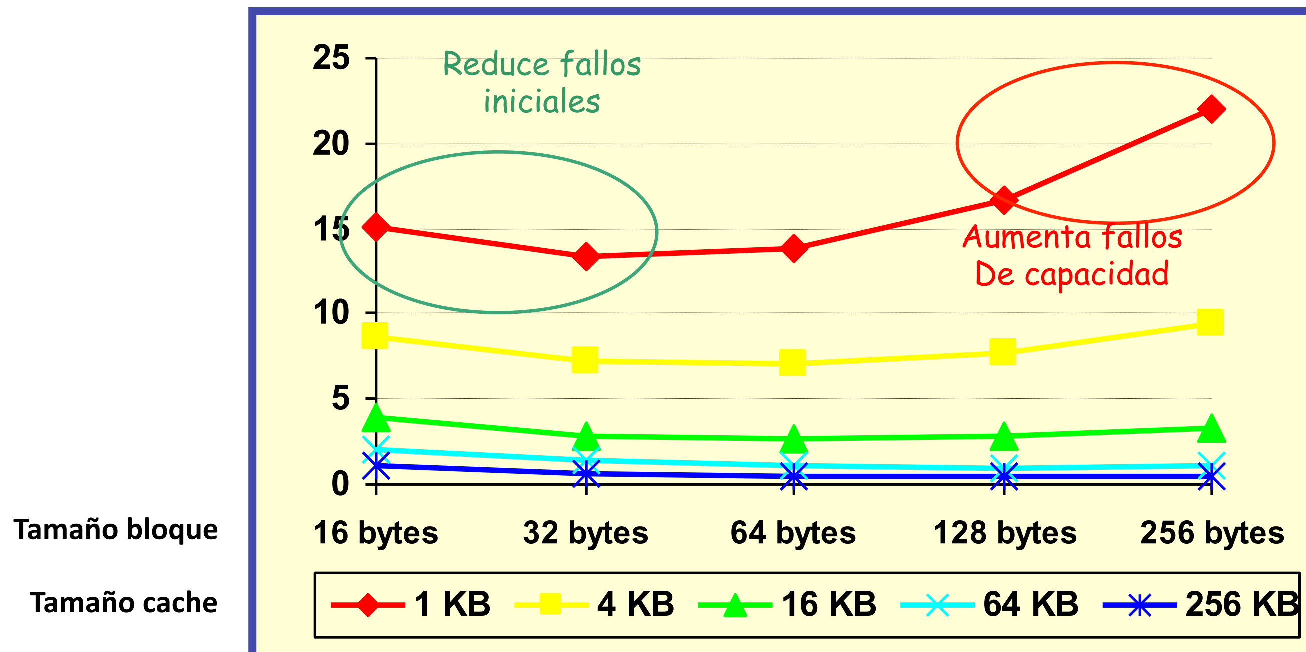
1-way
2-way
4-way
8-way
Capacity
Compulsory

1
2
4
8
16
32
64
128

Cache Size (KB)

¿Cómo afecta al rendimiento el tamaño de bloque?

- Aumentar el tamaño del bloque disminuye la tasa de fallos iniciales y **captura mejor la localidad espacial**
- Pero se aumenta la tasa de fallos de capacidad (menor Nº Bloques => **captura peor localidad temporal**)

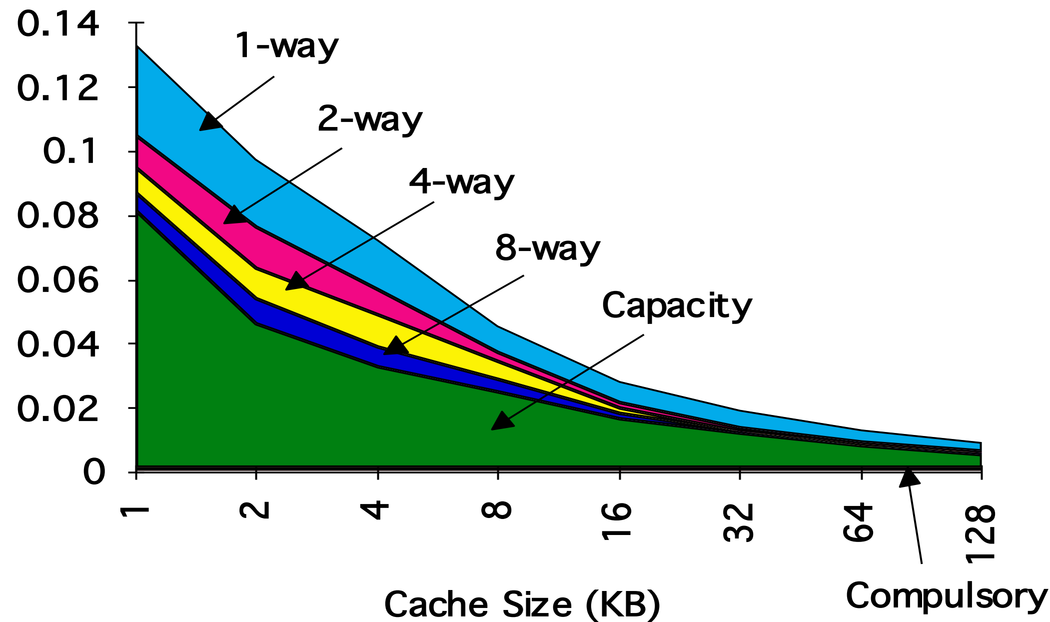


¿Cómo afectan al rendimiento el tamaño de cache y la asociatividad?

- **Tamaño de la cache:**
  - Aumentar el tamaño de la cache disminuye la tasa de fallos de capacidad.
  - Pero se aumentan el tiempo de acierto, el coste y el consumo

- **Asociatividad:**
  - El aumento de la asociatividad disminuye la tasa de fallos.
  - Pero se aumentan el tiempo de acceso y el hardware.

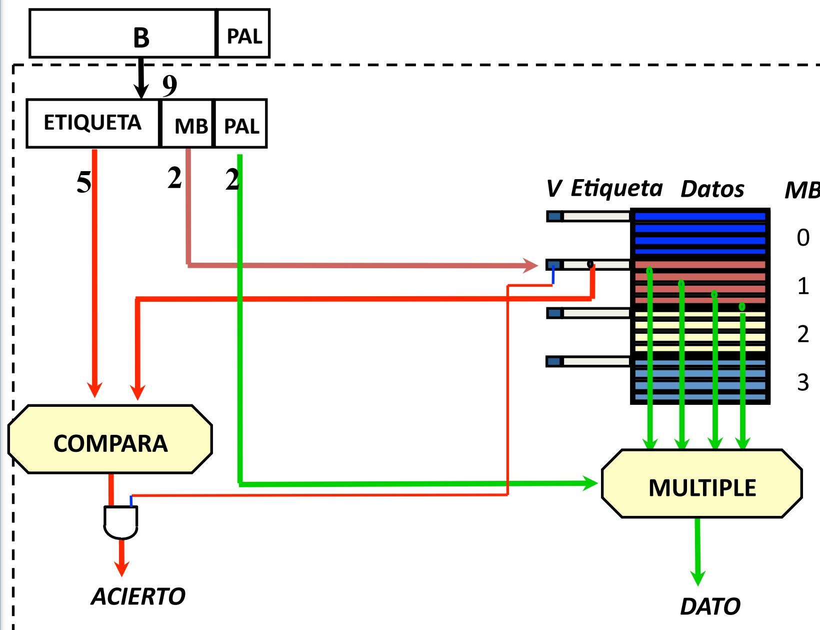
Tasa de fallos: Asociatividad



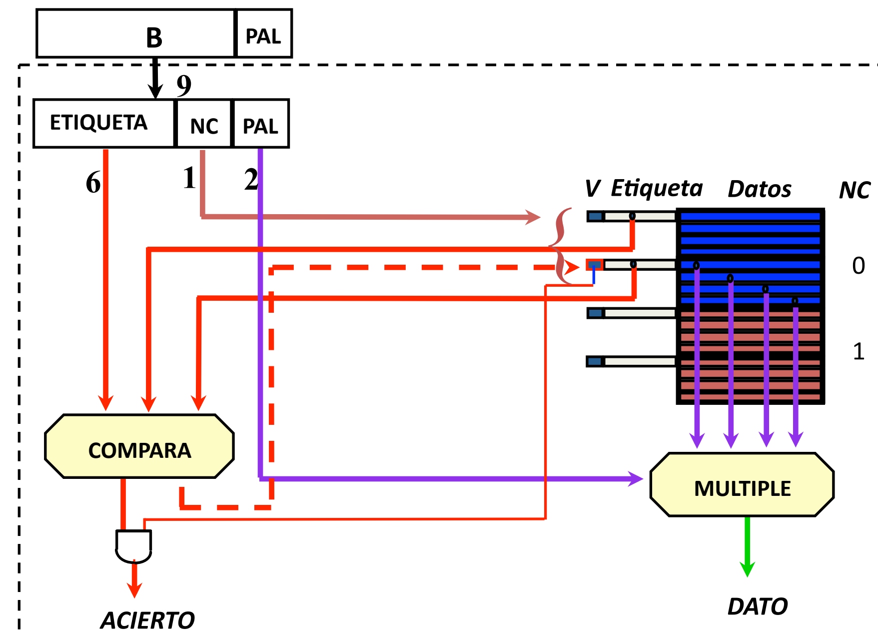
- El paso de emplazamiento directo a asociativa con 2 conjuntos permite disminuir el tamaño de la cache a la mitad
- Manteniendo el tamaño de la cache, no se consiguen grandes mejoras con más de 8 vías

Tiempo de acceso: cache L1 pequeña y sencilla

- El acceso al directorio y la comparación de tags consume tiempo
- Ejemplo: Comparación de acceso a un dato en cache directa y en cache asociativa por conjuntos con 2 vías



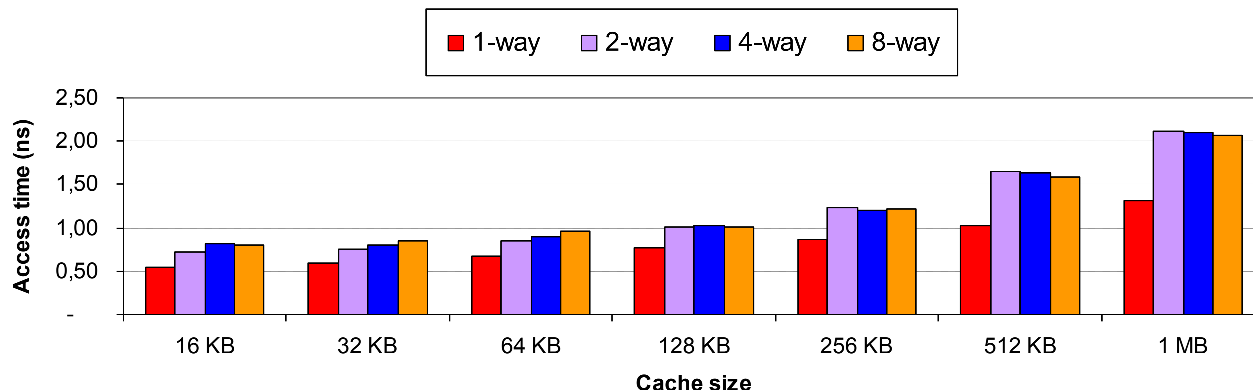
Directo



Asociativo por conjuntos 2 vías

Tiempo de acceso: cache L1 pequeña y sencilla

- **Caches simples y pequeñas**
  - Una cache pequeña se pueda integrar junto al procesador
    - evitando la penalización en tiempo del acceso al exterior
    - Tiempo de propagación versus tiempo de ciclo del procesador
  - Ejemplo: tres generaciones del procesadores AMD (K6, Athlon y Opteron) han mantenido el mismo tamaño para las caches L1
  - Simple (cache directa o grado de asociatividad pequeño)
    - En cache directa se puede solapar chequeo de tags y acceso al dato, puesto que el dato sólo puede estar en un lugar
    - El aumento del número de vías puede aumentar los tiempos de comparación de tags
  - Ejemplo: impacto del tamaño de la cache y la asociatividad sobre el tiempo de acceso (tecnología 90 nm)



¿Cómo mejorar el rendimiento de la cache?

Tcpu = N x [CPI + (AMPI x **Miss Rate** x **Miss Penalty**)] x **Tc**

- Estudio de técnicas para:
  - – Reducir la tasa de fallos
  - – Reducir la penalización del fallo
  - – Reducir el tiempo de acierto (hit time)
  - – Aumentar el ancho banda

¿Cómo mejorar el rendimiento de la cache?

Reducir tasa de fallos	Reducir penalización por fallo	Reducir tiempo de acierto	Aumentar ancho de banda
Tamaño de bloque Asociatividad Tamaño de Cache	Dar prioridad a las lecturas sobre las escrituras	Cache pequeña y sencilla	Cache no bloqueante
Optimización del código	Dar prioridad a la palabra crítica	Ocultar latencia de traducción DV => DF	Cache multibanco
Pre-búsqueda	Fusión de buffers de escritura	Predicción de vía	Cache segmentada
Caches víctimas	Caches multinivel	Cache de trazas	

Optimización de Código

- Para todos los ejemplos:
  - MC de 4 Kbytes
  - Acceso directo
  - 256 bloques
    - 1 bloque = 16 bytes
  - Datos:
    - int A[1024];   /*cada entero 4 bytes*/
    - int B[1024];
    - ¿Cuántas posiciones del array por bloque?
    - ¿Cuántas posiciones del array por MC?

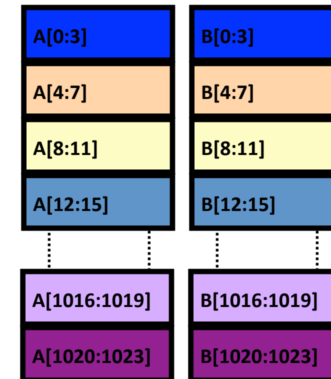
Optimización de Código

- Código original:

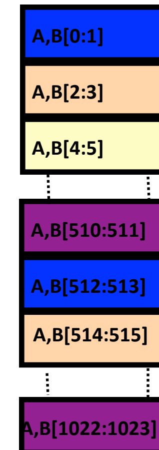
```
for (i = 0; i < 1024; i = i + 1)
        C = C + (A[i] + B[i]);
```

Si MC tuviera 2 vías, ¿se reducirían los fallos?

2x1024 accesos
2x1024 fallos
2x256 de inicio
Resto 2x3x256



- **Fusión de arrays**: Mejora la localidad espacial para disminuir los fallos
  - Colocar las mismas posiciones de diferentes arrays en posiciones contiguas de memoria




```
struct fusion{
        int A;
        int B;
} array[1024];
for (i = 0; i < 1024; i = i + 1)
        C = C + (array[i].A + array[i].B);
```

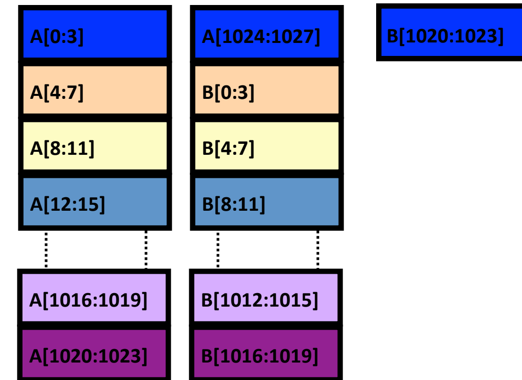
1024/2 fallos
2x256 de inicio

Ganancia: 4

Optimización de Código

- **Alargamiento de arrays**: Mejora la localidad espacial para disminuir los fallos
  - Impedir que en cada iteración del bucle se compita por el mismo marco de bloque

```
int A[1028];
int B[1024];
for (i=0; i < 1024; i=i+1)
            C = C + (A[i] + B[i]);
```



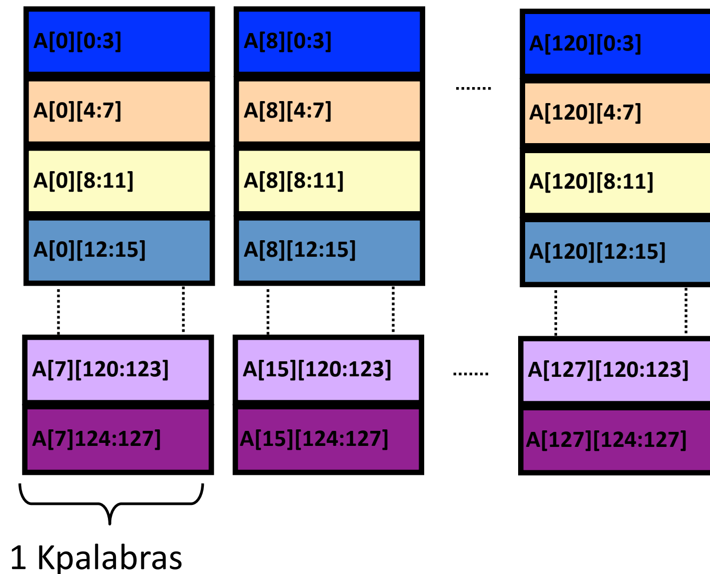
Si MC tuviera 2 vías, ¿se reducirían los fallos?

1024/2 fallos
2x256 de inicio

Ganancia: 4

Optimización de Código

- **Intercambio de bucles**: Mejora la localidad espacial para disminuir los fallos
  - En lenguaje C las matrices se almacenan por filas, luego se debe variar en el bucle interno la columna (int A[128][128];)



A tiene 2^{14} palabras = 16 Kpalabras => es **16 veces** mayor que MC

Optimización de Código

- **Intercambio de bucles:**
  - Acceso por filas

```
for (j=0; j < 128; j=j+1)
        for (i=0; i < 128;i=i+1)
                C = C * A[i][j];
```

128x128 accesos
128x128 fallos
16x256 de inicio
Resto (12288)

Si MC tuviera 2 vías, ¿se reducirían los fallos?

  - Acceso por columnas

```
for (i=0; i < 128;i=i+1)
        for (j=0; j < 128; j=j+1)
                C = C * A[i][j];
```

128x128/4 fallos
16x256 de inicio

Ganancia: 4

Optimización de Código

- **Fusión de bucles:**
  - Mejora la localidad temporal para disminuir los fallos
  - Fusionar los bucles que usen los mismos arrays para usar los datos que se encuentran en cache antes de desecharlos

```
for (i=0; i < 128; i=i+1))
        for (j=0; j < 128;j=j+1))
                C = C * A[i][j];
for (i=0; i < 128;i=i+1)
        for (j=0; j < 128;j=j+1)
                D = D + A[i][j];
```

128x128x2 accesos
(128x128/4)x2 fallos

```
for (i=0; i < 128; i=i+1))
        for (j=0; j < 128;j=j+1))
                C = C * A[i][j];
                D = D + A[i][j];
```

(128x128/4)x2 fallos

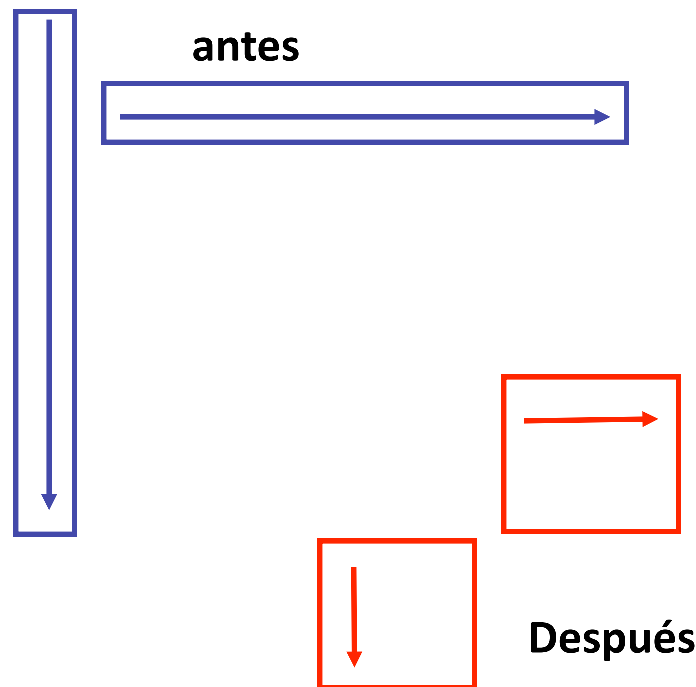
Ganancia: 4

¿Y sí aplicamos fusión de bucles e intercambio de bucles?

Optimización de Código: Blocking

- Mejora la **localidad temporal** para disminuir los **fallos de capacidad**

```
/* Antes */
for (i=0; i < N; i=i+1)
   for (j=0; j < N; j=j+1)
        {r = 0;
         for (k=0; k < N; k=k+1)
            r = r + y[i][k]*z[k][j];
         x[i][j] = r;
        };
```



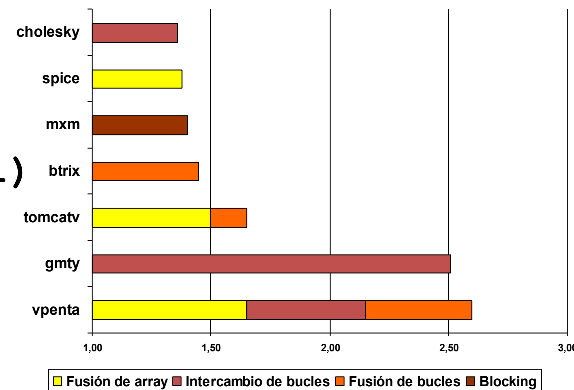
- ✓ Dos bucles internos. Para cada valor de i:
  - Lee todos los NxN elementos de **z**
  - Lee N elementos de 1 fila de **y**
  - Escribe N elementos de 1 fila de **x**
- ✓ Fallos de capacidad dependen de N y Tamaño de la cache:
- ✓ Idea: calcular por submatrices BxB que permita el tamaño dela cache

Optimización de Código: Blocking

Mejora la **localidad temporal** para disminuir los **fallos de capacidad**

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

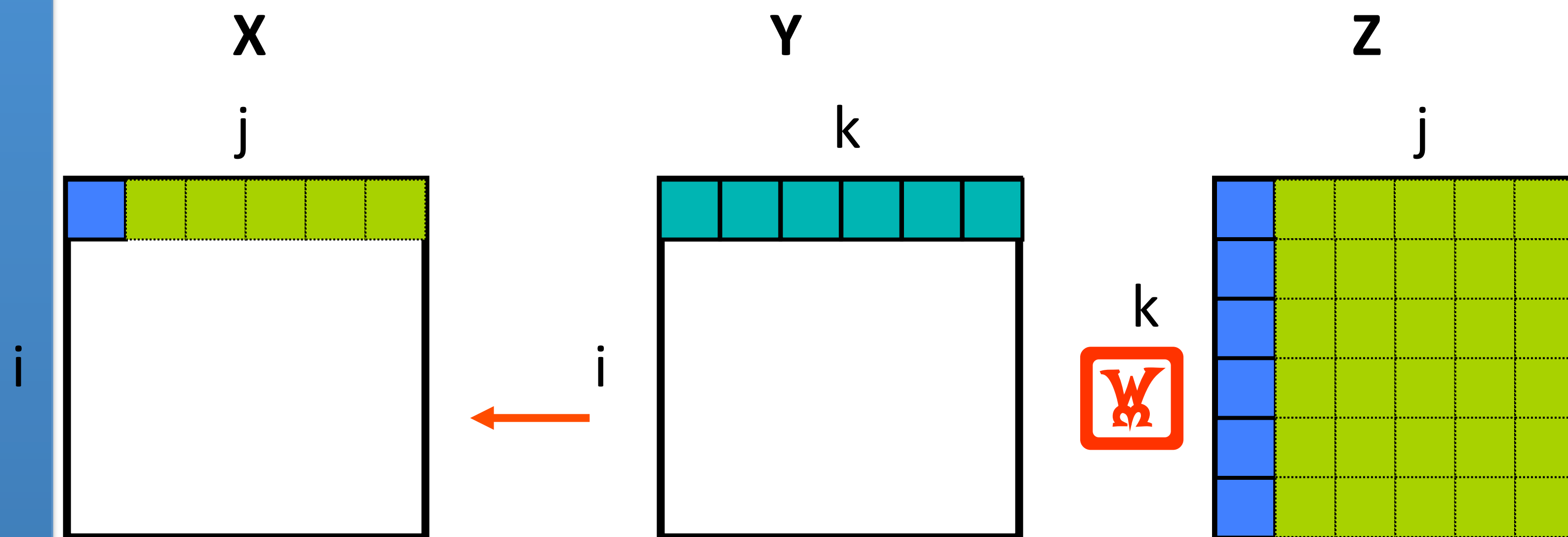
Mejora de rendimiento



- ✓ B *Factor de bloque (Blocking Factor)*

Optimización de Código: Blocking

- Ejemplo: Producto de matrices 6x6 (sin blocking)



i = 0, j = 0, k = 0..5

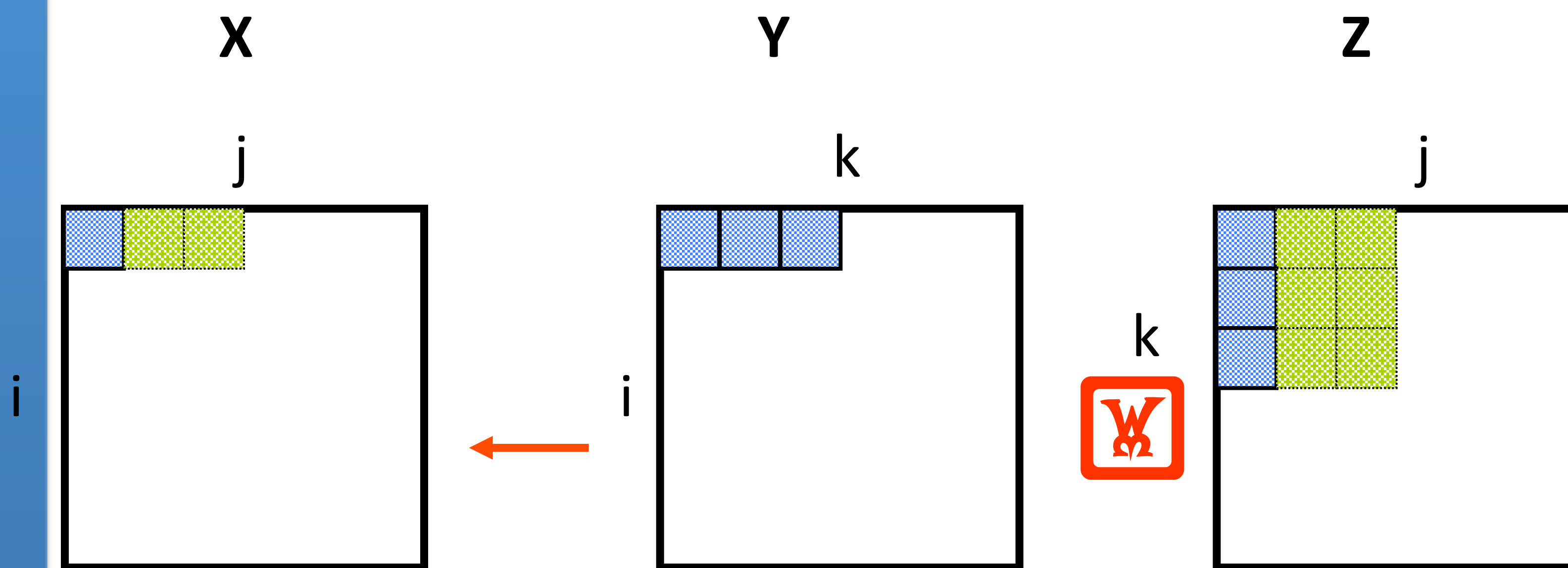
$$X_{ij} = \sum_k Y_{ik} Z_{kj}$$

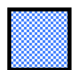
i = 0, j = 1..5 , k = 0..5

Al procesar la 2ª fila de Y (i=1) se necesita de nuevo 1ª col de Z: ¿Está todavía en la cache? Cache insuficiente provoca múltiples fallos sobre los mismos datos

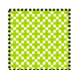
Optimización de Código: Blocking

- Ejemplo “blocking”: Con Blocking (B=3)





i = 0, j = 0, k = 0..2

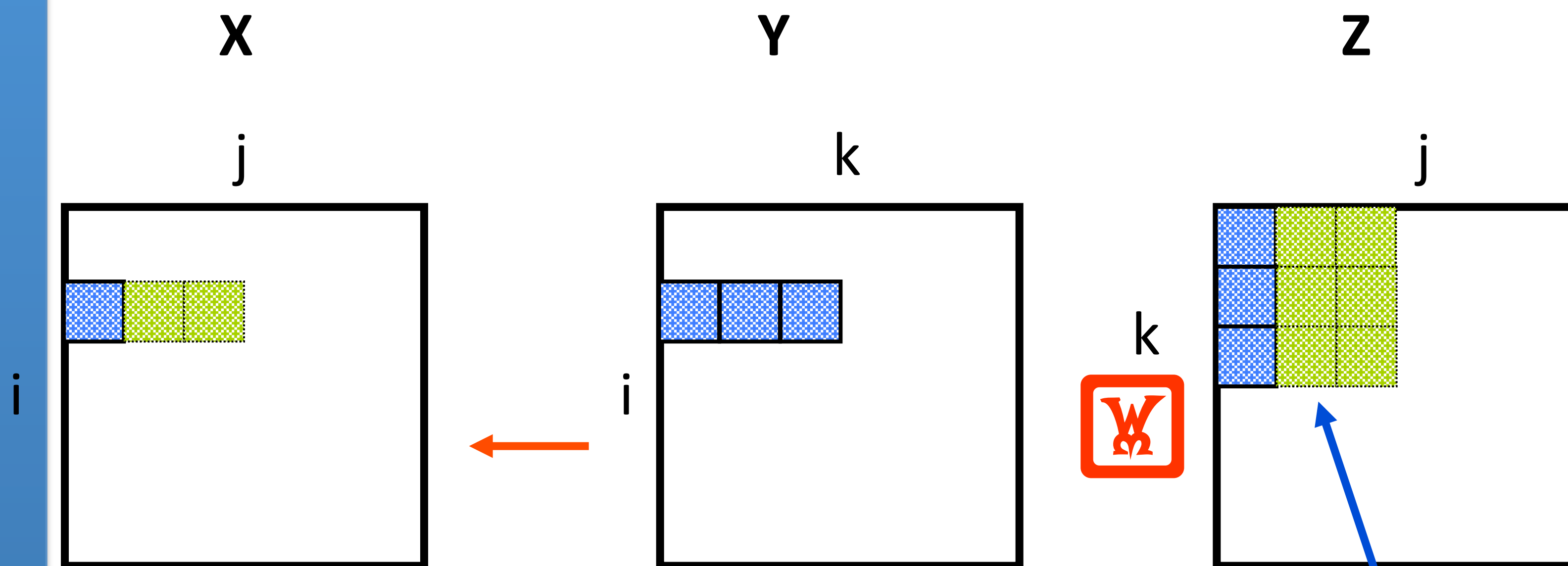


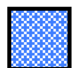
i = 0, j = 1..2 , k = 0..2

Evidentemente, los elementos de X no están completamente calculados

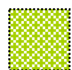
Optimización de Código: Blocking

- Ejemplo "blocking": Con Blocking (B=3)





i = 1, j = 0, k = 0..2

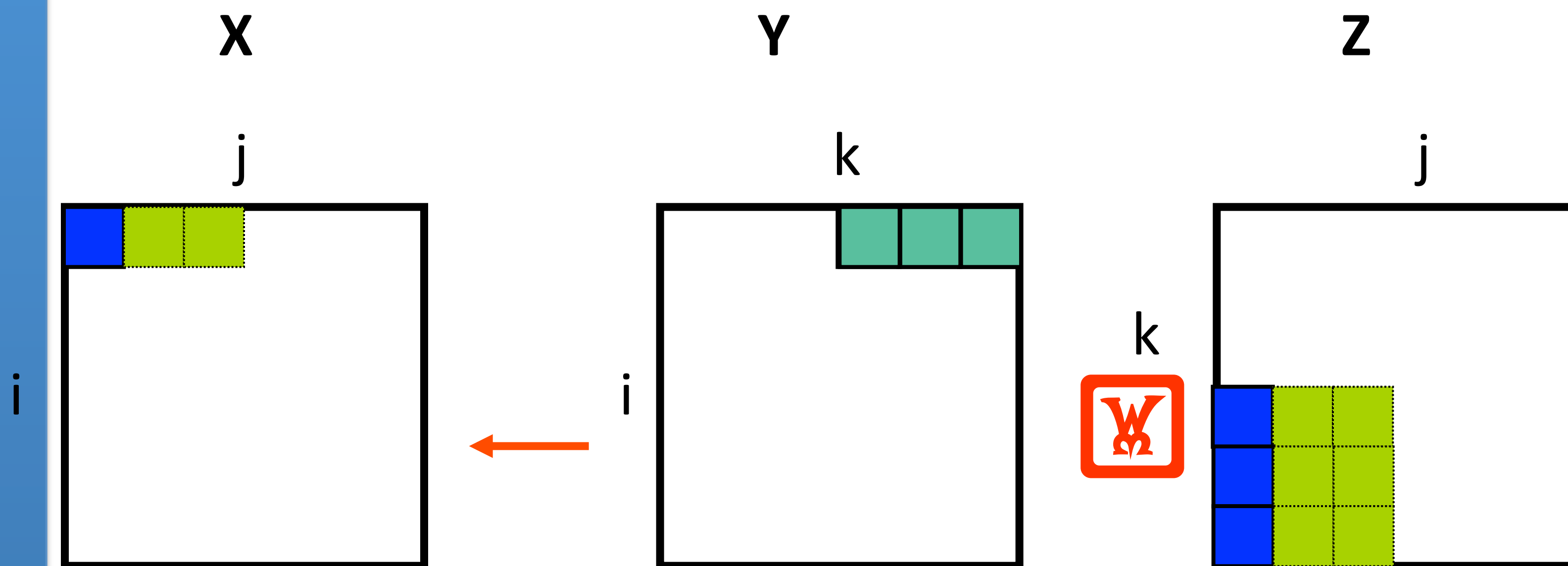


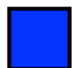
i = 1, j = 1..2 , k = 0..2

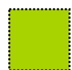
Idea: Procesar el bloque 3x3 de Z antes de quitarlo de la cache

Optimización de Código: Blocking

- Con Blocking (B=3). Algunos pasos después...



 i = 0, j = 0, k = 3..5

 i = 0, j = 1..2 , k = 3..5

Y ya empezamos a tener elementos de X completamente calculados!

¿Cómo mejorar el rendimiento de la cache?

Reducir tasa de fallos	Reducir penalización por fallo	Reducir tiempo de acierto	Aumentar ancho de banda
Tamaño de bloque Asociatividad Tamaño de Cache	Dar prioridad a las lecturas sobre las escrituras	Cache pequeña y sencilla	Cache no bloqueante
Optimización del código	Dar prioridad a la palabra crítica	Ocultar latencia de traducción DV => DF	Cache multibanco
Pre-búsqueda	Fusión de buffers de escritura	Predicción de vía	Cache segmentada
Caches víctimas	Caches multinivel	Cache de trazas	

Cache con Pre-búsqueda

- Reduce los fallos de cache anticipando las búsquedas antes de que el procesador demande el dato o la instrucción que provocarían un fallo
  - Esta técnica reduce la tasa de fallos
  - Este tipo de cache sólo es efectiva con un adecuado ancho de banda ya que se van a producir más transferencias de bloques de las necesarias
  - Esta técnica es más efectiva si se dispone de una memoria cache y de un buffer, en caso contrario los bloques pre-buscados (que puede que no es usen) están desalojando de MC bloques activos

Cache con Pre-búsqueda

- Pre-búsqueda Hardware:
  - Típicamente: la CPU busca dos bloques en un fallo (el referenciado y el siguiente)
  - El bloque buscado se lleva a MC
  - El pre-buscado se lleva a un buffer ("prefetch buffer" o "stream buffer"). Al ser referenciado pasa a MC

Cache con Pre-búsqueda

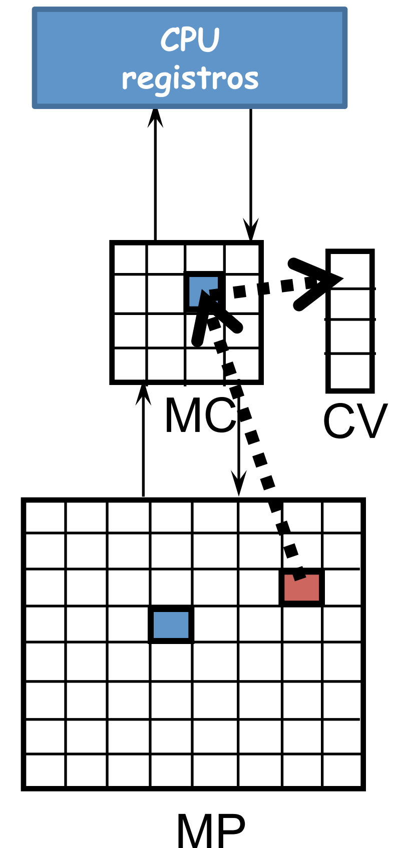
- Pre-búsqueda Software:
  - Instrucciones especiales de pre-búsqueda introducidas por el compilador
  - La eficiencia depende del compilador y del tipo de programa
  - Pre-búsqueda con destino en cache (MIPS IV, PowerPC, SPARC v. 9), o en registro (HP-PA)
  - Instrucciones de pre-búsqueda no producen excepciones. Es una forma de especulación.
  - Funciona bien con bucles y patrones simples de acceso a arrays. Aplicaciones de cálculo
  - Funciona mal con aplicaciones enteras que presentan un amplio reuso de cache
  - Overhead por las nuevas instrucciones. Más búsquedas. Más ocupación de memoria

¿Cómo mejorar el rendimiento de la cache?

Reducir tasa de fallos	Reducir penalización por fallo	Reducir tiempo de acierto	Aumentar ancho de banda
Tamaño de bloque Asociatividad Tamaño de Cache	Dar prioridad a las lecturas sobre las escrituras	Cache pequeña y sencilla	Cache no bloqueante
Optimización del código	Dar prioridad a la palabra crítica	Ocultar latencia de traducción DV => DF	Cache multibanco
Pre-búsqueda	Fusión de buffers de escritura	Predicción de vía	Cache segmentada
Caches víctimas	Caches multinivel	Cache de trazas	

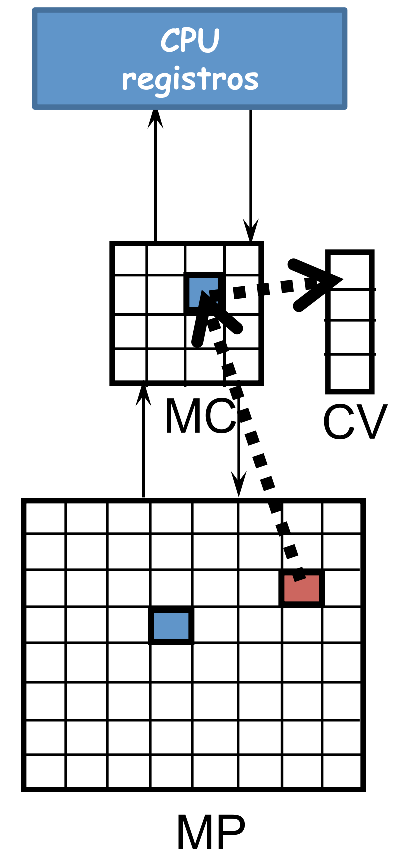
Cache Víctima

- Se propuso por primera vez en 1990, la idea es utilizar una cache pequeña completamente asociativa (CV) conjuntamente con una cache grande con una asociatividad más limitada (MC)
- Se intenta conseguir que MC tenga la misma tasa de fallos que si fuera completamente asociativa
  - Ej: MC de acceso directo + CV asociativa tiene la misma tasa de fallos que MC con 2 vías



Cache Víctima

- Un bloque puede estar en MC o en CV pero nunca en las dos
- En la cache víctima estarán aquellos bloques que han sido re-emplazados de la memoria cache
  - Un bloque pasa de MC a CV cuando se produce un re-emplazo
  - Un bloque pasa de CV a MC cuando es referenciado

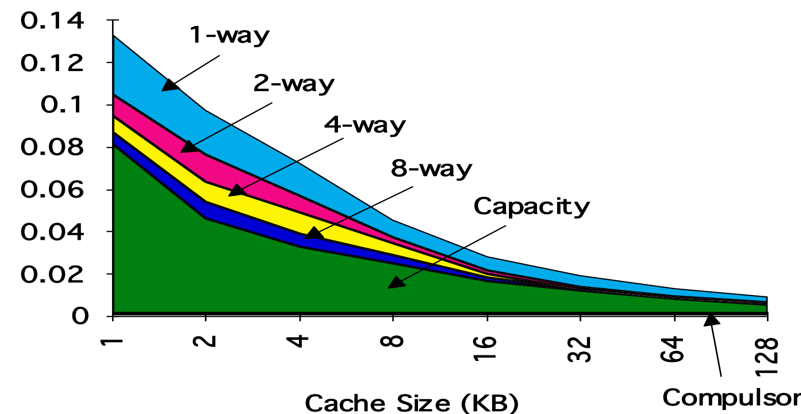


¿Cómo mejorar el rendimiento de la cache?

Reducir tasa de fallos	Reducir penalización por fallo	Reducir tiempo de acierto	Aumentar ancho de banda
Tamaño de bloque Asociatividad Tamaño de Cache	Dar prioridad a las lecturas sobre las escrituras	Cache pequeña y sencilla	Cache no bloqueante
Optimización del código	Dar prioridad a la palabra crítica	Ocultar latencia de traducción DV => DF	Cache multibanco
Pre-búsqueda	Fusión de buffers de escritura	Predicción de vía	Cache segmentada
Caches víctimas	Caches multinivel	Cache de trazas	

Caches Multinivel

- El aumento del tamaño de MC reduce la tasa de fallos
- El aumento del tamaño de la MC aumenta la penalización por fallo
- El aumento del tamaño de MC aumenta el coste del procesador
- Si se quiere una MC rápida y barata, ésta tiene que ser pequeña
- Teniendo diferentes niveles de MC se puede conseguir reducir la penalización por fallo
  - No es lo mismo transferir de MP a MC que de L1 a L2
  - Tener L2 permite aplicar políticas de pre-búsqueda con menos inconvenientes



Cache Multinivel

- **Cache multinivel (L2, L3, …)**
  - Tiempo medio de acceso a memoria (TMAM): Un nivel
    - TMAM = Hit time + Miss Rate x Miss Penalty
  - Tiempo medio de acceso a memoria: Dos niveles

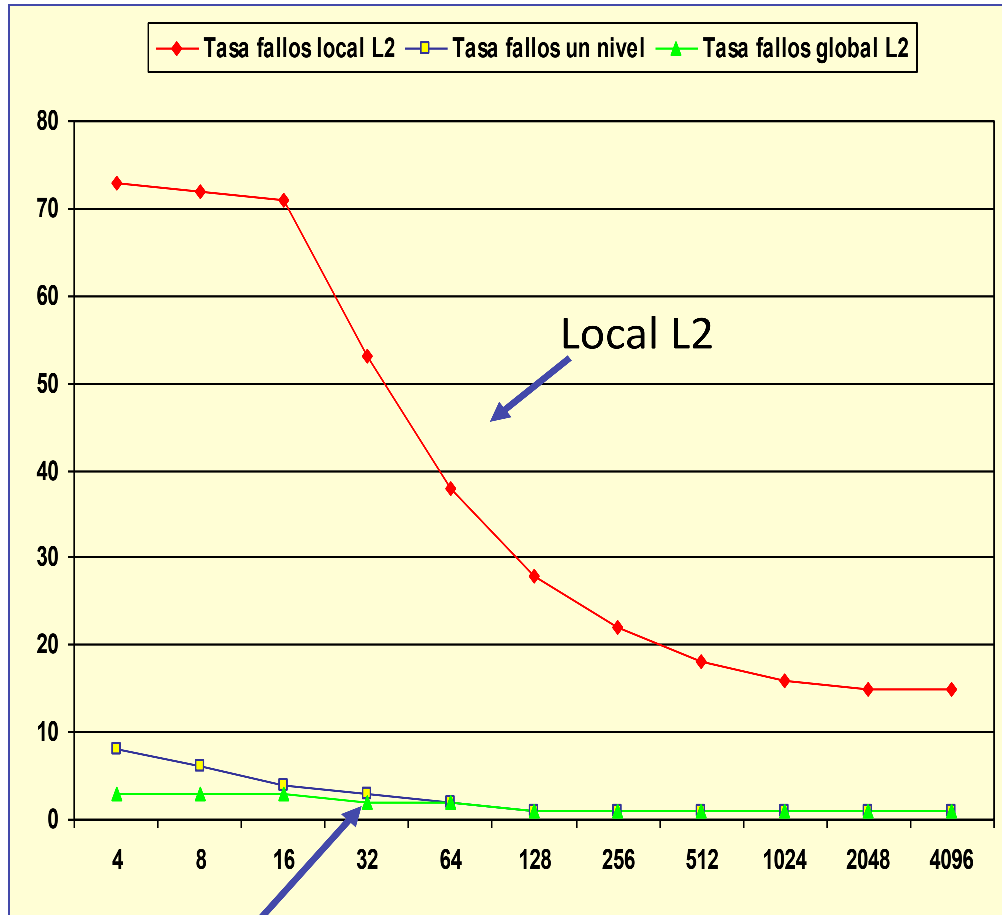
    $TMAM = Hit\ Time_{L1} + Miss\ Rate_{L1} \times Miss\ Penalty_{L1}$

    $Miss\ Penalty_{L1} = Hit\ Time_{L2} + Miss\ Rate_{L2} \times Miss\ Penalty_{L2}$

    $\Rightarrow TMAM = Hit\ Time_{L1} + Miss\ Rate_{L1} \times [Hit\ Time_{L2} + (Miss\ Rate_{L2} \times Miss\ Penalty_{L2})]$

  - Definiciones:
    - Tasa de fallos local en una cache (Lx): fallos en cache Lx dividido por el número total de accesos a la cache Lx
    - Tasa de fallos global en una cache (Lx): fallos en cache Lx dividido por el número total de accesos a memoria generados por el procesador
  - Consecuencia: Tasa de fallos local en L1 = Tasa de fallos global en L1
    - La tasa de fallos global es lo importante
    - L1: Afecta directamente al procesador => Acceder a un dato en el ciclo del procesador
    - L2: Afecta a la penalización de L1 => Reducción del tiempo medio de acceso

Cache Multinivel



¿Cómo mejorar el rendimiento de la cache?

Reducir tasa de fallos	Reducir penalización por fallo	Reducir tiempo de acierto	Aumentar ancho de banda
Tamaño de bloque Asociatividad Tamaño de Cache	Dar prioridad a las lecturas sobre las escrituras	Cache pequeña y sencilla	Cache no bloqueante
Optimización del código	Dar prioridad a la palabra crítica	Ocultar latencia de traducción DV => DF	Cache multibanco
Pre-búsqueda	Fusión de buffers de escritura	Predicción de vía	Cache segmentada
Caches víctimas	Caches multinivel	Cache de trazas	

Reducción penalización por fallo

- **Dar prioridad a las lecturas sobre las escrituras**
  - Un fallo de lectura puede impedir la continuación de la ejecución del programa; un fallo de escritura puede ocultarse
  - Buffer de escrituras (rápido). Depositar en buffer las palabras que tienen que ser actualizadas en MP y continuar ejecución.
  - La transferencia del buffer a MP se realiza en paralelo con la ejecución del programa
  - PROBLEMA: podemos estar intentando leer una palabra que todavía está en el buffer

¿Cómo mejorar el rendimiento de la cache?

Reducir tasa de fallos	Reducir penalización por fallo	Reducir tiempo de acierto	Aumentar ancho de banda
Tamaño de bloque Asociatividad Tamaño de Cache	Dar prioridad a las lecturas sobre las escrituras	Cache pequeña y sencilla	Cache no bloqueante
Optimización del código	Dar prioridad a la palabra crítica	Ocultar latencia de traducción DV => DF	Cache multibanco
Pre-búsqueda	Fusión de buffers de escritura	Predicción de vía	Cache segmentada
Caches víctimas	Caches multinivel	Cache de trazas	

Reducción penalización por fallo

- **<u>Dar prioridad a palabras críticas</u>**
  - Cuando la palabra solicitada se carga en memoria cache se envía al procesador, sin esperar a la carga del bloque completo
  - Problema. Localidad espacial: alta probabilidad de acceder a continuación a la siguiente palabra en secuencia.

¿Cómo mejorar el rendimiento de la cache?

Reducir tasa de fallos	Reducir penalización por fallo	Reducir tiempo de acierto	Aumentar ancho de banda
Tamaño de bloque Asociatividad Tamaño de Cache	Dar prioridad a las lecturas sobre las escrituras	Cache pequeña y sencilla	Cache no bloqueante
Optimización del código	Dar prioridad a la palabra crítica	Ocultar latencia de traducción DV => DF	Cache multibanco
Pre-búsqueda	Fusión de buffers de escritura	Predicción de vía	Cache segmentada
Caches víctimas	Caches multinivel	Cache de trazas	

¿Cómo mejorar el rendimiento de la cache?

Reducir tasa de fallos	Reducir penalización por fallo	Reducir tiempo de acierto	Aumentar ancho de banda
Tamaño de bloque Asociatividad Tamaño de Cache	Dar prioridad a las lecturas sobre las escrituras	Cache pequeña y sencilla	Cache no bloqueante
Optimización del código	Dar prioridad a la palabra crítica	Ocultar latencia de traducción DV => DF	Cache multibanco
Pre-búsqueda	Fusión de buffers de escritura	Predicción de vía	Cache segmentada
Caches víctimas	Caches multinivel	Cache de trazas	

Cache Segmentada

- Segmentar los accesos a la cache permite aumentar el ancho de banda.
- Problema: incremento de los ciclos de latencia. Más ciclos de reloj entre el lanzamiento de un LD y el uso de los datos que el LD proporciona
- Más problemas: Mayor penalización en los saltos mal predichos
- Ejemplos: Nº de etapas del acceso a la cache en diferentes procesadores
  - Pentium 1 etapa
  - De Pentium Pro a Pentium III 2 etapas
  - Pentium 4 4 etapas

Cache Segmentada: ejemplo

Address calculation | Disambiguation | Cache access | Result drive

Address calculation

offset

index

...

Address decoder

Data Array

Way mux

Aligner

Data

Address decoder

Tag Array

=

Hit/Miss

Processor Microarchitecture. An implementation Perspective. A. González et al. Morgan¬Claypool Publ. 2011.

Cache Resumen (I)

Técnica	Tasa fallos	Penal fallo	Tiempo acierto	Ancho banda	Coste HW / Complejidad	Comentario
Aumento tamaño de bloque	+	-			0	Trivial. L2 de Pentium 4 usa 128 bytes
Aumento asociatividad	+		-		1	Ampliamente usado
Aumento tamaño de MC	+		-		1	Ampliamente usado, especialmente en L2
Optimización del compilador	+				0	El software presenta oportunidades de mejora. Algunos computadores tienen opciones de optimización
Prebúsqueda HW	+	+			2 instr., 3 data	Muchos procesadores prebuscan instrucciones. AMD Opteron y Pentium 4 prebuscan datos.
Prebúsqueda SW	+	+			3	Necesita cache no bloqueante. En muchas CPUs.

Cache Resumen (II)

Técnica	Tasa fallos	Penal fallo	Tiempo acierto	Ancho banda	Coste HW / Complejidad	Comentario
Prioridad a las lecturas		+			1	Ampliamente usado
Prioridad a la palabra crítica		+			2	Ampliamente usado
Cache multinivel		+			2	Ampliamente usado. Más complejo si tamaño de bloque en L1 y L2 distintos.
Cache pequeña y sencilla	–		+		0	Trivial; ampliamente usado.
Cache multibanco				+	1	En L2 de Opteron y Niagara
Cache segmentada			–	+	1	Ampliamente usado