Memoria Cache

Águstín Fernández, Josep Llosa, Fermín Sánchez

Estructura de Computadors II
Departament d'Arquitectura de Computadors
Facultat d'Informàtica de Barcelona

Visión General de la Jerarquía

- La jerarquía de Memorias está justificada por las siguientes situaciones:
  - Velocidad Procesadores >> Velocidad Memorias.
  - Propiedades de las Memorias.
  - Propiedades de los Programas.

Velocidad Procesadores >> Velocidad Memorias

Año	Procesador	Frec.	Número máximo referencias/ciclo	Ancho Banda necesario
1991	DEC Alpha 21064	200 MHz	2.5	1.90 Gbytes/s
2000	DEC Alpha 21264	600 MHz	5	11.44 Gbytes/s
2003	Intel Pentium 4	3 GHz	3.75	42.75 Gbytes/s

Año	Tipo Memoria	Capacidad	Frec. Placa Base	Latencia 1r dato / resto	Ancho Banda lect. 32 bytes
2000	EDO DRAM	64 Mbits	66 MHz	50 / 20 ns	193.9 Mbytes/s
2002	SDRAM	128 Mbits	167 MHz	36 / 6 ns	592.5 Mbytes/s
2003	DDR SDRAM	256 Mbits	200 MHz	30 /2.5 ns	853.3 Mbytes/s

Propiedades de las Memorias

Tipo	Capacidad	Tiempo acceso	Tecnología	Coste por Mbyte
Registros	64 – 1024 bytes	0.3 – 1ns	-	-
Memoria Cache	8Kb – 2 Mb	1 - 5 ns	Semiconductor SRAM	6.75 €
Memoria Principal	1Mb – 1 Gb	10 – 30 ns	Semiconductor DRAM	0.25 €
Memoria Secundaria	10 Gb – 200 Gb	10 – 50 ms	Disco Duro Magnética	0.00125 €

↑**Capacidad** ⇔ ↑Tiempo de acceso ⇔ ↓**Coste por Mbyte**

↓Capacidad ⇔ ↓**Tiempo de acceso** ⇔ ↑Coste por Mbye

Datos de febrero de 2003

Visión General de la Jerarquía

- Propiedades de los Programas:
  - Regla del 90/10
  - Localidad Temporal
  - Localidad Espacial

Propiedades de los Programas

- Regla del 90 / 10
  - El 90% de todas las referencias a memoria (datos e instrucciones) son realizadas por el 10% del código.
  - Spec2000

%@	% referencias a datos	% referencias a Instrucciones
1%	79,6%	90,7%
2%	84,8%	97,3%
5%	90,4%	99,6%
10%	93,7%	100,0%

El 2% de las posiciones de memoria de datos reciben el 84,8% de todas las referencias a datos.

El 5% de todas las instrucciones reciben el 99,6% de todas las referencias a instrucciones.

Propiedades de los Programas

- Localidad Temporal
  - Si accedemos a una posición de memoria, es muy probable que se vuelva a acceder a la misma posición en un futuro cercano.

```
void Xrut(int v[], int w[])
{ int vert, polig, color;
  . . .
  for (i=0; i<5000; i++) {
    v[i] = w[i] | 0x01;
    v[i] = v[i] * vert;
    if (v[i] != 0)
      w[i] = polig / v[i];
    v[i] = w[i] + color;
  }
  . . .
}
```

¿Por qué? Por los bucles.

•Instrucciones: dentro de un bucle se accede repetidamente a las mismas instrucciones.

•Datos: dentro de un bucle se accede repetidamente a las mismas variables.

Propiedades de los Programas

- Localidad Espacial
  - Si accedemos a una posición de memoria, es muy probable que se acceda a posiciones próximas en un futuro cercano.

```
void Xrut(int v[], int w[])
{ int vert, polig, color;
  . . .
  for (i=0; i<5000; i++) {
    v[i] = w[i] | 0x01;
    v[i] = v[i] * vert;
    if (v[i] != 0)
      w[i] = polig / v[i];
    v[i] = w[i] + color;
  }
  . . .
}
```

¿Por qué?

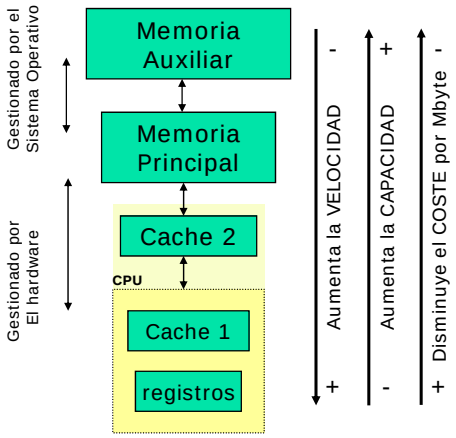
•Instrucciones: las instrucciones se ejecutan en secuencia.

•Datos: los vectores y matrices se suelen recorrer completos; los parámetros y variables locales están en el bloque de activación de la subrutina.

Propiedades de los Programas

- ¿Cómo podemos aprovechar la localidad?
  - Localidad Temporal: si traemos un dato (o instrucción) de memoria, sería útil guardarlo "cerca" del procesador para que los futuros accesos sean más rápidos.
  - Localidad Espacial: si traemos un dato (o instrucción) de memoria, sería útil traer también los datos próximos y dejarlos "cerca" del procesador. Esto sólo tiene sentido si traer datos próximos sólo cuesta un poco más que traer un solo dato.

Solución: Jerarquía de Memorias



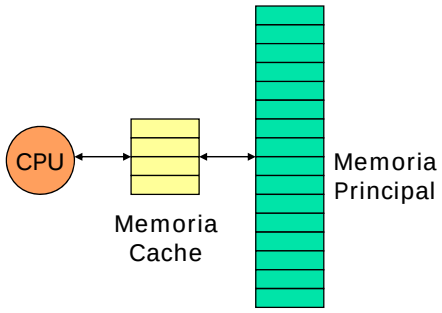
Objetivo: que cuando el procesador acceda a un dato, éste se encuentre en los niveles de la jerarquía más cercanos al procesador.

Si se consigue, obtendríamos una memoria con la velocidad de los niveles rápidos, el tamaño de los niveles más grandes y todo ello con un coste razonable.

¡La jerarquía funciona por las propiedades (localidad) de los programas!

Principios de Funcionamiento de las Memorias Cache

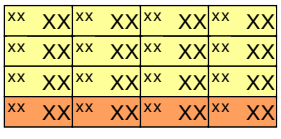
- Memoria Cache: memoria pequeña y rápida que almacena una parte del contenido de una memoria más grande y lenta. La memoria cache se encargará de que la información que se almacene sea útil.



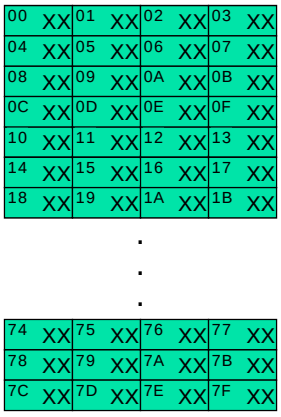
- Objetivo:
  - Velocidad de la memoria cache.
  - Capacidad de la memoria principal.
  - Coste de la memoria principal más un porcentaje razonable.
- Esta solución es posible debido a la localidad de los programas. La cache retiene información recientemente usada e información próxima a la recientemente usada.

Principios de Funcionamiento de las Memorias Cache

La memoria cache (MC) es una memoria de acceso rápido organizada en líneas (bloques)

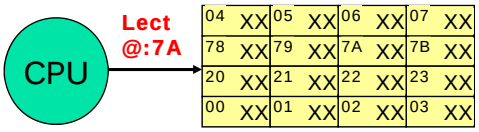


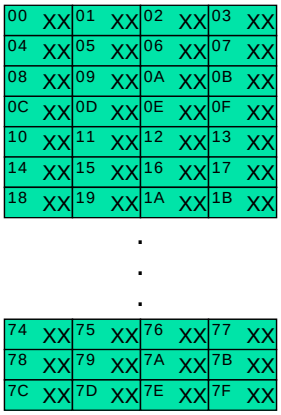
Línea de cache



Principios de Funcionamiento de las Memorias Cache

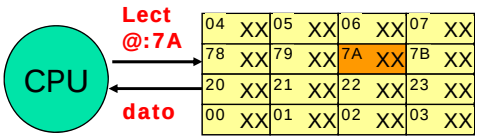
Cuando la CPU realiza un acceso a memoria, en primer lugar se busca el dato en la memoria cache.

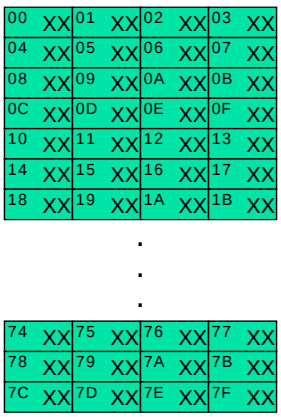




Principios de Funcionamiento de las Memorias Cache

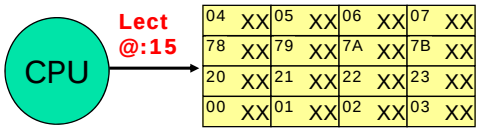
Si el dato buscado está en la MC se produce un **ACIERTO** (**hit**) y se pasa el dato a la CPU de forma rápida.

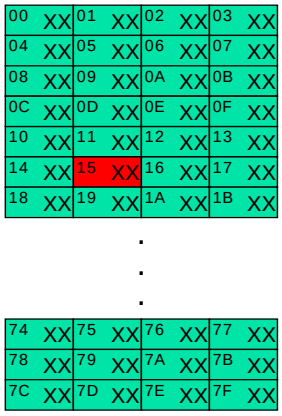




Principios de Funcionamiento de las Memorias Cache

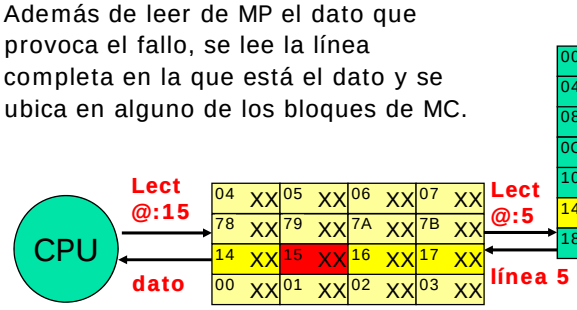
Si el dato buscado NO está en la MC se produce un **FALLO** (**miss**) y se ha de buscar en la MP.

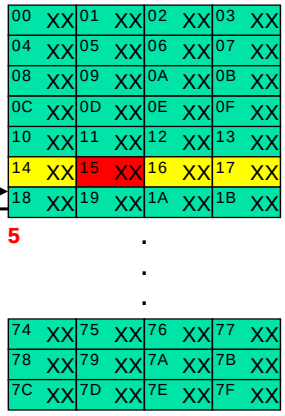


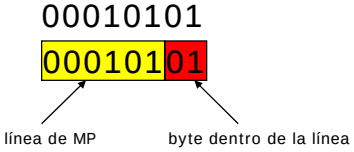


Principios de Funcionamiento de las Memorias Cache

Además de leer de MP el dato que provoca el fallo, se lee la línea completa en la que está el dato y se ubica en alguno de los bloques de MC.







Principios de Funcionamiento de las Memorias Cache

- La Memoria Principal (MP) se organiza en líneas.
- Cada línea es un conjunto de bytes almacenados en posiciones consecutivas.
- La memoria cache (MC) es una memoria de acceso rápido organizada el bloques de tamaño 1 línea.
- Cuando la CPU realiza un acceso a memoria, en primer lugar se busca en la memoria cache.
- Si el dato buscado está en la MC se produce un **ACIERTO** (**hit**) y se pasa el dato a la CPU de forma rápida.
- Si el dato buscado **NO** está en la MC se produce un **FALLO** (**miss**) y se ha de leer de la MP.
- Además de leer de MP el dato que provoca el fallo, se lee la línea completa en la que está el dato y se ubica en alguno de los bloques de MC.
- La MC es transparente al programador.
- La información se transfiere de MP a MC (y viceversa) de forma automática a medida que el procesador va generando accesos a memoria.

Principios de Funcionamiento de las Memorias Cache

- En cualquier Memoria Cache hay que definir:
  - Algoritmo de Emplazamiento: determina en qué bloques de MC puede colocarse una línea. Determina, también, dónde hay que buscar un dato.
  - Algoritmo de reemplazo: determina qué línea se ha de eliminar de la cache para dejar espacio a una nueva línea.
  - Políticas de escritura: determina cómo se hacen las escrituras. En cualquier caso, al final siempre se ha de escribir en MP.
- Han de ser algoritmos hardware:
  - Algoritmos sencillos.
  - Algoritmos rápidos.

Medidas de Rendimiento

- Tasa de Aciertos
  h = #aciertos / #referencias
- Tasa de Fallos
  m = #fallos / #referencias = 1-h

¿De qué dependen?
- Del tamaño de Cache
- Del tamaño de Línea
- De los algoritmos de Emplazamiento y Reemplazo
- Del Programa evaluado

Medidas de Rendimiento

- Coste en ciclos de un acceso
  - Tiempo de servicio en caso de acierto: tsa
  - Tiempo de servicio en caso de fallo: tsf = tsa + tpf
  - Tiempo de penalización en caso de fallo: tpf
  - Tiempo medio de acceso a memoria:
    Tma = h·tsa + m·tsf = tsa + m·tpf

¡Atención! Esta formulación sólo es aplicable en MCs de sólo lectura porque no tiene en cuenta políticas de escritura.

Medidas de Rendimiento

- Impacto de la MC:
  - Tiempo de ejecución de un programa:

    Tejec = N · CPI · Tc

    N: instrucciones ejecutadas

    CPI: ciclos por instrucción en media

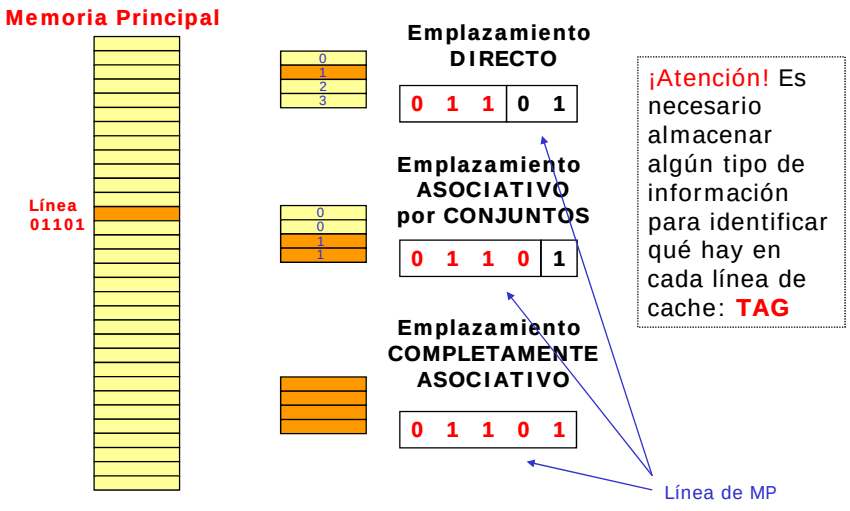
    Tc: tiempo de ciclo
    - CPI = CPIideal + CPImem
    - CPImem son los ciclos que perdemos por tener una cache imperfecta ($m \neq 0$).

      CPImem = nr . (Tma – tsa)

      CPImem = nr · m · tpf (caso particular de una MC de sólo lectura)

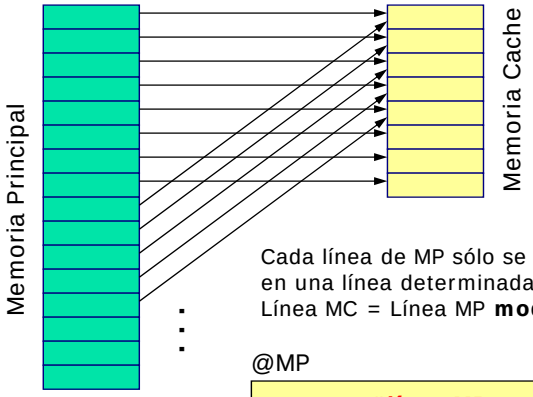
(nr: número de referencias por instrucción)

Algoritmos de Emplazamiento



¡Atención! Es necesario almacenar algún tipo de información para identificar qué hay en cada línea de cache: **TAG**

Emplazamiento Directo



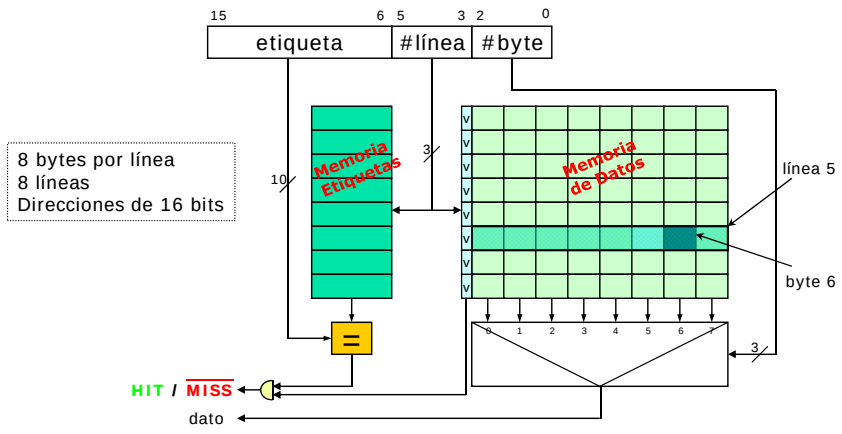
Cada línea de MP sólo se puede ubicar en una línea determinada de MC:
Línea MC = Línea MP **mod** #líneas MC

@MP

#línea MP	#byte

TAG	#línea MC	#byte

Cache Directa



Evaluación Cache Directa

- Es la cache que tiene MENOR tiempo de acceso (tsa): se accede simultáneamente a la memoria de datos y de etiquetas.
- Es la cache que tiene MAYOR tasa de fallos (m).

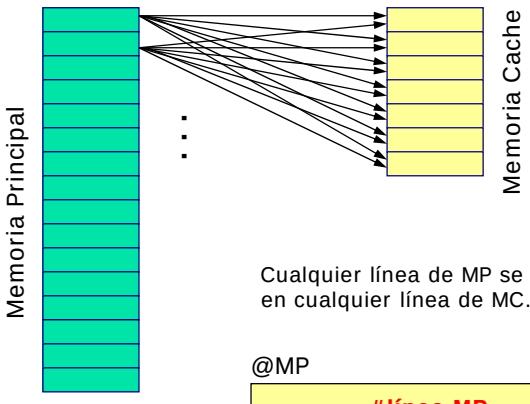
```
for (i=0; i<N; i++)
    sum = sum+v[i];
```

Código con localidad espacial.
Si cada elemento de v ocupa 4 bytes y las líneas son de 32 bytes, la tasa de fallos es 0,125 (se repite el patrón 1 fallo y 7 aciertos).

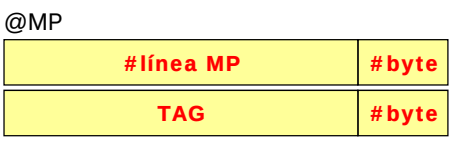
```
for (i=0; i<N; i++)
    sum = sum+v[i]+w[i];
```

Código con localidad espacial.
Si los elementos v[i] y w[i] van a parar a la misma línea de cache, la tasa de fallos es 1 (todo son fallos).

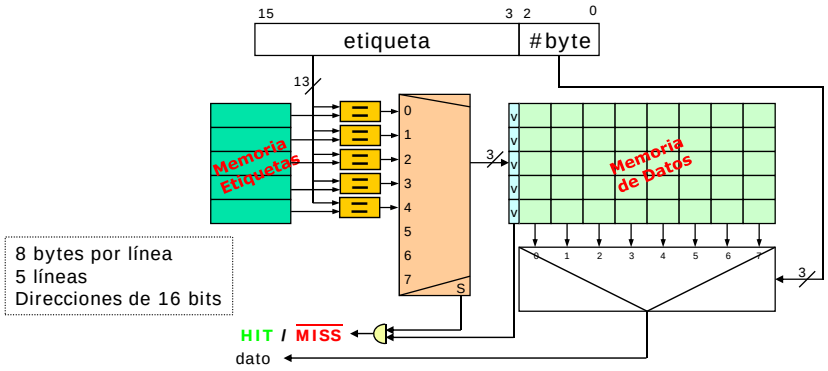
Emplazamiento Completamente Asociativo



Cualquier línea de MP se puede ubicar en cualquier línea de MC.



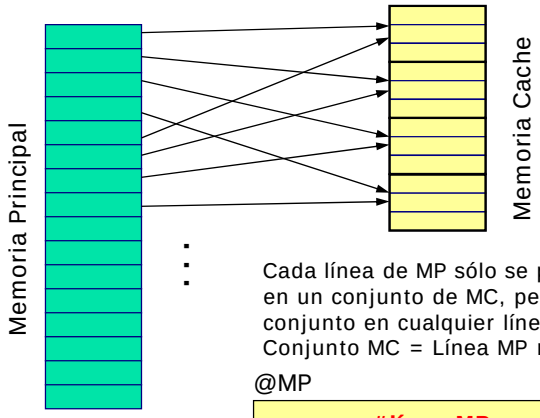
Cache Completamente Asociativa



Evaluación Cache Completamente Asociativa

- Es la cache que tiene MAYOR tiempo de acceso (tsa): primero se accede a la memoria de etiquetas y a continuación (en caso de acierto) se accede a la memoria de datos.
- Es la cache que tiene MENOR tasa de fallos (m).
  - En este caso, no se pueden producir fallos por conflicto de datos porque cualquier línea de MP se puede ubicar en cualquier línea de la cache.
  - Los fallos son de capacidad.

Emplazamiento Asociativo por Conjuntos



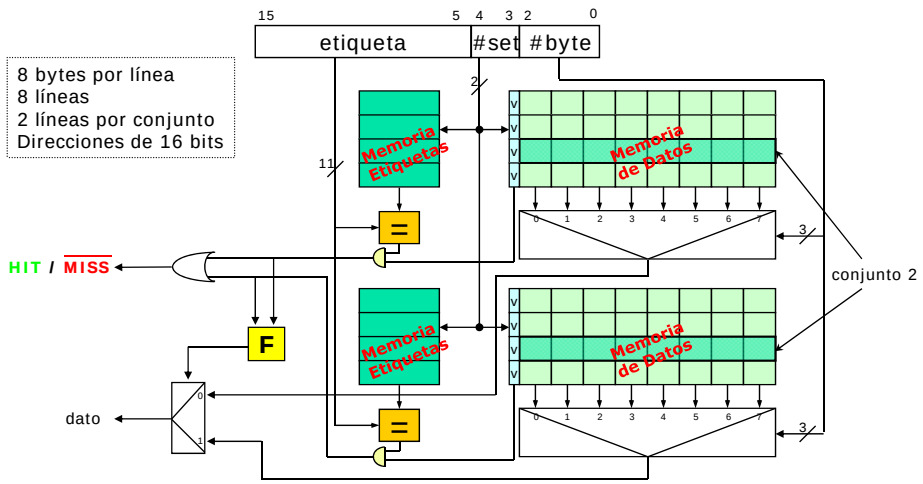
Cada línea de MP sólo se puede ubicar en un conjunto de MC, pero dentro del conjunto en cualquier línea:
Conjunto MC = Línea MP **mod** #Conjuntos

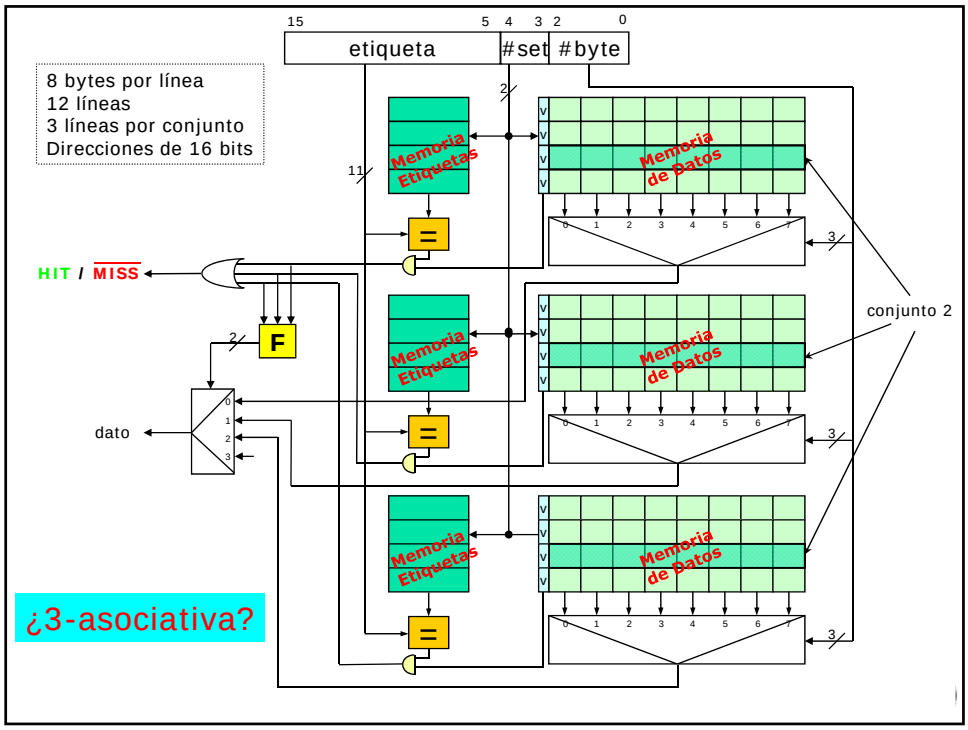
@MP

#línea MP		#byte
TAG	#conjunto MC	#byte

Cache Asociativa por Conjuntos

8 bytes por línea
8 líneas
2 líneas por conjunto
Direcciones de 16 bits





Evaluación Cache Asociativa por Conjuntos

- Esta cache tiene MAYOR tiempo de acceso (tsa) que una MC directa. El coste adicional viene provocado por el multiplexor utilizado para seleccionar entre las vías del conjunto.
- Esta cache tiene MENOR tasa de fallos (m) que una MC directa, pero MAYOR que una cache completamente asociativa.

```
for (i=0; i<N; i++)
   sum = sum+v[i]+w[i];
```

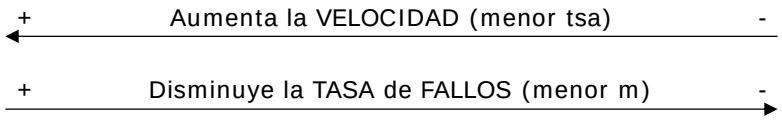
Este código no tendría ningún conflicto en una cache 2-asociativa.

```
for (i=0; i<N; i++)
   … v[i]+w[i]+x[i] … ;
```

Sin embargo, este código tendría problemas de conflicto en una cache 2-asociativa si v[i], w[i] y x[i] van al mismo conjunto de cache.

Comparación 3 algoritmos emplazamiento

Memoria Cache DIRECTA	MC Asociativa por Conjuntos	MC Completamente Asociativa



- MC Directa: MC Asociativa por Conjuntos con 1 línea por conjunto.
- MC Completamente Asociativa: MC Asociativa por conjuntos con 1 único conjunto.
- ¡Atención! Para poder comparar correctamente las diversas configuraciones de cache hay que utilizar otros parámetros, como por ejemplo, el tiempo medio de acceso (Tma).

Problema MC

- Disponemos de un sistema de Memoria con direcciones de 32 bits. En este sistema queremos evaluar 3 alternativas:
  - Una MC directa de 64Kbytes y 32 bytes por línea.
  - Una MC asociativa por conjuntos de 96 Kbytes, con 3 líneas por conjunto y 64 bytes por línea.
  - Una MC completamente asociativa de 16 Kbytes y líneas de 256 bytes.
- Para cada una de estas 3 alternativas, se pide lo siguiente:
  1. Dada una dirección de memoria indicad, mediante un dibujo, qué bits se utilizan para seleccionar el byte dentro de la línea, qué bits se utilizan para seleccionar el conjunto (o línea) de memoria cache, y qué bits forman la etiqueta.
  2. ¿Cuántos bits ocupa la memoria de etiquetas (tags)?

Algoritmos de Reemplazo

- Si se produce un fallo y el conjunto donde debe ubicarse la nueva línea está lleno,

  ¿Qué línea del conjunto hay que reemplazar?

- ¿Algoritmo importante? Un comportamiento deficiente puede provocar que reemplacemos una línea que se va a utilizar en un acceso próximo.
- Algoritmos hardware muy simples. Se han de ejecutar en un plazo de tiempo muy pequeño.
- Los algoritmos de reemplazo se aplican en caso de fallo, pero algunos de ellos necesitan actualizar cierta información después de cada acierto.
- En una MC Directa no tiene sentido hablar de algoritmo de reemplazo.

Algoritmos de Reemplazo

- Reemplazo Aleatorio
  - Se selecciona aleatoriamente una línea de entre todas las candidatas a ser reemplazadas.
  - Es un algoritmo muy sencillo de implementar.
  - A pesar de su aparente falta de sentido, funciona bastante bien.

Algoritmos de Reemplazo

- Reemplazo FIFO (First In First Out)
  - De entre todas las líneas candidatas a ser reemplazadas, se selecciona la que lleva más tiempo en la cache.
  - Es un algoritmo muy sencillo de implementar, sólo es necesario utilizar un contador con módulo.
  - Tiene un comportamiento patológico no deseable porque no tiene en cuenta la utilización.
    - Existen secuencias de referencias a memoria que provocan más fallos con N+1 líneas por conjunto que con N líneas. Lo deseable es que al aumentar el tamaño de la MC disminuya el número de fallos.

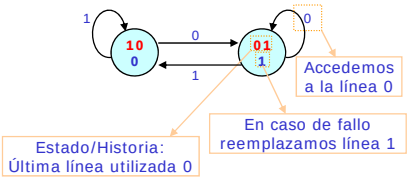
Algoritmos de Reemplazo

- Reemplazo LRU (Least Recently Used)
  - De entre todas las líneas candidatas a ser reemplazadas, se selecciona la que lleva más tiempo en la cache sin ser utilizada.
  - Este algoritmo da buenos resultados. Teniendo en cuenta el comportamiento de los programas, parece la opción más lógica.
  - Sin embargo, es muy costoso de implementar si el grado de asociatividad es alto. El coste de implementar este algoritmo es n! (siendo n el grado de asociatividad).
  - Normalmente se implementa un algoritmo PseudoLRU. Un algoritmo LRU ha de mantener información de en qué orden se ha accedido a todas las líneas de un conjunto. En un algoritmo pseudoLRU sólo se mantiene parte de esa información.

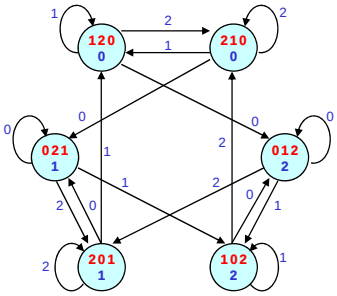
Algoritmos de Reemplazo

- Posible Implementación LRU:
  - Un registro de estado para cada conjunto. En función del estado sabemos qué línea hay que reemplazar.

2-asociativa



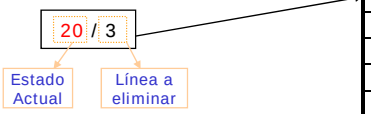
3-asociativa



Implementación imposible para un número elevado de líneas por conjunto (n). Número de estados = n!

Algoritmos de Reemplazo

- Posible Implementación pseudoLRU cache 5-asociativa
  - Si tenemos n líneas por conjunto, sólo mantenemos información de las k últimas líneas utilizadas.



Una cache 5-asociativa necesitaría 120 estados para implementar el algoritmo LRU completo.

5-asoc	0	1	2	3	4
01 / 2	01	10	20	30	40
02 / 3	02	10	20	30	40
03 / 4	03	10	20	30	40
04 / 1	04	10	20	30	40
10 / 2	01	10	21	31	41
12 / 3	01	12	21	31	41
13 / 4	01	13	21	31	41
14 / 0	01	14	21	31	41
20 / 3	02	12	20	32	42
21 / 4	02	12	21	32	42
23 / 0	02	12	23	32	42
24 / 1	02	12	24	32	42
30 / 4	03	13	23	30	43
31 / 0	03	13	23	31	43
32 / 1	03	13	23	32	43
34 / 2	03	13	23	34	43
40 / 1	04	14	24	34	40
41 / 2	04	14	24	34	41
42 / 3	04	14	24	34	42
43 / 0	04	14	24	34	43

Algoritmos de Reemplazo

% fallos	go				tomcatv				ijpeg			
	Random	FIFO	Pseudo LRU	LRU	Random	FIFO	Pseudo LRU	LRU	Random	FIFO	Pseudo LRU	LRU
2 Kbytes	22,4	21,8	20,2	19,6	24,3	23,0	24,1	23,1	9,09	8,51	8,79	8,08
4 Kbytes	13,2	12,7	11,6	11,1	23,2	21,6	23,3	22,2	6,75	6,51	6,47	6,28
8 Kbytes	6,80	6,28	5,73	5,34	22,7	21,0	23,0	21,7	5,13	5,09	5,09	5,02
16 Kbytes	3,22	2,88	2,66	2,49	22,2	20,7	22,4	21,4	1,73	1,62	1,60	1,31
32 Kbytes	1,59	1,44	1,32	1,25	12,1	8,53	9,40	9,18	0,66	0,61	0,52	0,49
64 Kbytes	0,55	0,53	0,44	0,42	8,05	6,40	6,57	6,55	0,41	0,40	0,36	0,36
128 Kbytes	0,07	0,08	0,06	0,05	6,73	6,37	6,37	6,37	0,15	0,18	0,15	0,16

A la vista de estos resultados, se puede concluir que los algoritmos de reemplazo no influyen sustancialmente en el rendimiento de la MC.

Políticas de Escritura

- Premisa: las escrituras, finalmente, se han de hacer en Memoria Principal.
- Problemas a resolver:
  - ¿Cuándo se actualiza la MP?
  - ¿Qué hacemos cuando se produce un fallo en MC por una escritura?

Políticas de Escritura

¿Cuándo se actualiza la Memoria Principal?:

- WRITE THROUGH (escritura a través o escritura inmediata)
  - Se actualizan simultáneamente la MC y la MP.
  - El tiempo de servicio es el tiempo de acceso a MP.
  - La MP siempre está actualizada.
  - Se puede reducir el tiempo de escritura utilizando buffers.
- COPY BACK (escritura diferida)
  - En una escritura sólo se actualiza MC.
  - Para cada línea se añade un bit de control (*dirty bit*) que indica si la línea ha sido modificada o no.
  - Cuando una línea ha de ser reemplazada, entonces se actualiza la MP (sólo si la línea ha sido previamente modificada).
  - Las escrituras son rápidas (velocidad de MC).
  - El tiempo de penalización en caso de fallo aumenta.
  - Durante un tiempo existe una inconsistencia entre MP y MC.

Políticas de Escritura

¿Qué hacer en caso de fallo en escritura?

- WRITE ALLOCATE (con migración en caso de fallo)
  - Se trae la línea de MP a MC y después se realiza la escritura.
- WRITE NO ALLOCATE (sin migración en caso de fallo)
  - La línea NO se trae a MC. Esto obliga a realizar la escritura directamente en MP.

Políticas de Escritura

- Normalmente se combina:
  - WRITE THROUGH con WRITE NO ALLOCATE
  - COPY BACK con WRITE ALLOCATE
  - Otras combinaciones son teóricamente posibles.
  - Problemas: 1-11

Problema

- Disponemos de un procesador de 16 bits con bus de direcciones de 24 bits. La memoria cache tiene las siguientes características:
  - Emplazamiento directo
  - Tamaño total: 4096 bytes
  - Tamaño de línea: 16 bytes
  - Política de escritura: copy back + write allocate

Rellenad la siguiente tabla:

Problema

Tipo	Dirección Hexa	Línea MP	Línea MC	Hit/Miss	Lectura de MP			Escritura en MP		
					si/no	Dirección	Tamaño	si/no	Dirección	Tamaño
R byte	ECA130									
W word	ECA131									
W byte	EC2172									
W word	ECA133									
R byte	EC3174									
R word	EC3175									
R byte	ECB136									
W word	ECA137									
R byte	EC2178									
R word	ECB139									

Políticas de Escritura (Tma). Problema

Se quiere diseñar la memoria cache para un determinado procesador.

Se barajan dos alternativas:

a) Con escritura inmediata y sin carga en caso de fallo de escritura.

b) Con escritura retardada y carga en caso de fallo de escritura

Se han obtenido por simulación las siguientes medidas:

- porcentaje de escrituras: 20%
- porcentaje de líneas modificadas: 33.33 %
- tasa de aciertos caso (a): 0.9
- tasa de aciertos caso (b): 0.85

El tiempo de acceso a memoria cache es de 10 ns y el tiempo de memoria principal para escribir una palabra es de 80 ns. Para leer o escribir una línea en la memoria principal se emplean 100 ns.

Se pide:

1) Calculad el tiempo invertido en ejecutar 1000 accesos para las dos alternativas. Detallad el número de accesos de cada tipo y el tiempo empleado para cada uno de ellos.
2) Indicad qué alternativa sería la más rápida para un programa que sólo realizara lecturas.
3) Indicad qué motivos pueden existir para que la escritura de una palabra tarde ligeramente menos que la escritura de una línea.

Políticas de Escritura (Tma)

- COPY BACK + WRITE ALLOCATE

$T_{ma} = h \cdot t_{sa} + m \cdot (\%lineas_{MOD} \cdot tsf_{MOD} + \%lineas_{NOMOD} \cdot tsf_{NOMOD})$

siendo:

- $\%líneas_{MOD}$: porcentaje de líneas modificadas
- tsf_{MOD}: tiempo de servicio en caso de fallo cuando se reemplaza una línea modificada (sucia).
- $\%líneas_{NOMOD}$: porcentaje de líneas NO modificadas
- tsf_{NOMOD}: tiempo de servicio en caso de fallo cuando se reemplaza una línea NO modificada (limpia).

Políticas de Escritura (Tma)

- WRITE THROUGH + WRITE NO ALLOCATE

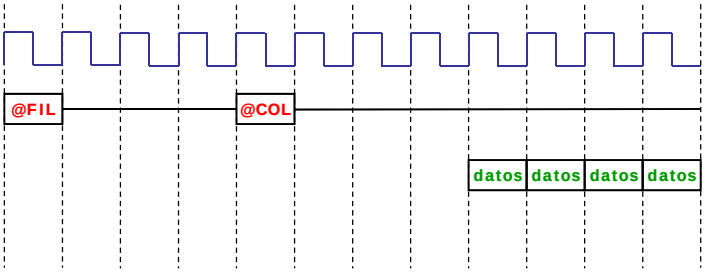
$T_{ma} = \%escr \cdot ts_{WR} + \%lect \cdot (h \cdot tsa_{RD} + m \cdot tsf_{RD})$

siendo:

- %escr: porcentaje de escrituras
- %lect: porcentaje de lecturas
- ts_{WR}: tiempo de servicio de una escritura en MP.
- tsa_{RD}: tiempo de servicio de una lectura en caso de acierto
- tsf_{RD}: tiempo de servicio de una lectura en caso de fallo

Memoria Entrelazada (repaso)

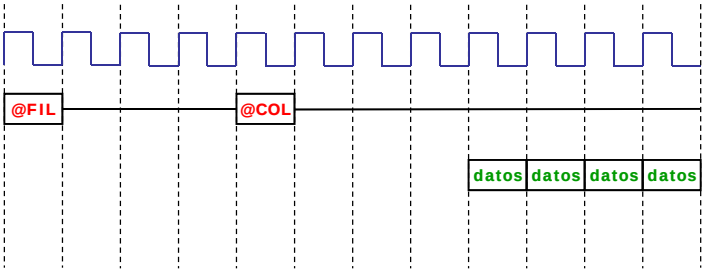
- Lectura de 1 línea de 32 bytes, la MP está organizada en DIMMs de 8 bytes de ancho.
  - Cronograma Simplificado:
    - Latencia fila (4 ciclos), latencia columna (4 ciclos), velocidad transferencia (8 bytes por ciclo)



  - La velocidad de salida / transferencia de los datos dependerá del tipo de Memoria y de la placa base (buses)

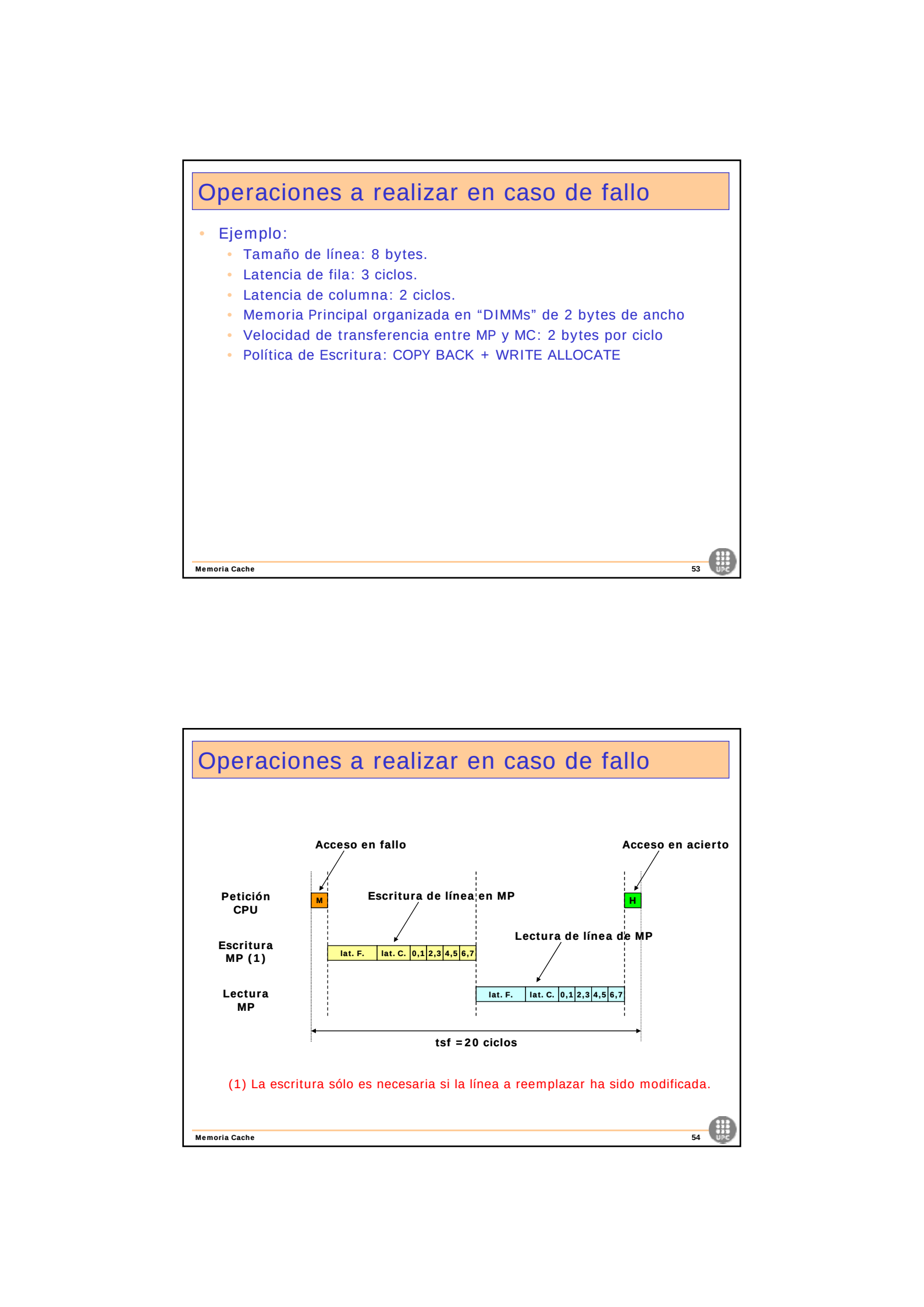
Memoria Entrelazada (repaso)

- Escritura de 1 línea de 32 bytes, la MP está organizada en DIMMs de 8 bytes de ancho.
  - Cronograma Simplificado:
    - Latencia fila (4 ciclos), latencia columna (4 ciclos), velocidad transferencia (8 bytes por ciclo)



Operaciones a realizar en caso de fallo

- Ejemplo:
  - Tamaño de línea: 8 bytes.
  - Latencia de fila: 3 ciclos.
  - Latencia de columna: 2 ciclos.
  - Memoria Principal organizada en “DIMMs” de 2 bytes de ancho
  - Velocidad de transferencia entre MP y MC: 2 bytes por ciclo
  - Política de Escritura: COPY BACK + WRITE ALLOCATE

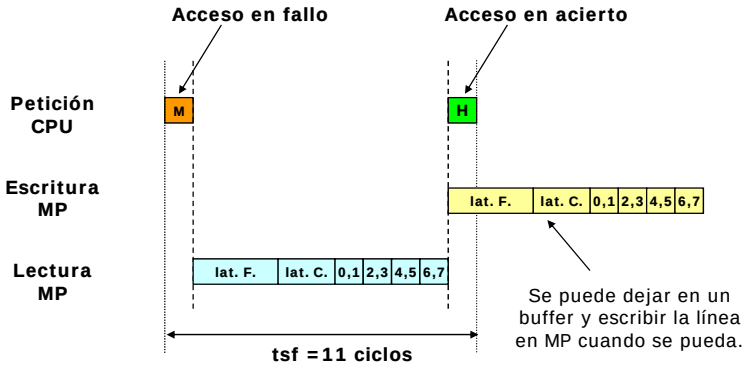


Operaciones a realizar en caso de fallo

(1) La escritura sólo es necesaria si la línea a reemplazar ha sido modificada.

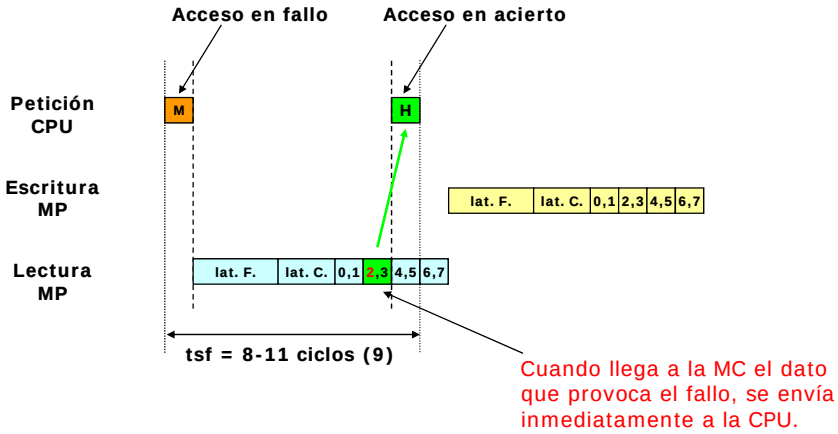
Operaciones a realizar en caso de fallo

- Reducción del tiempo de fallo: en una MC con COPY BACK: actualizar MP después de leer la línea.



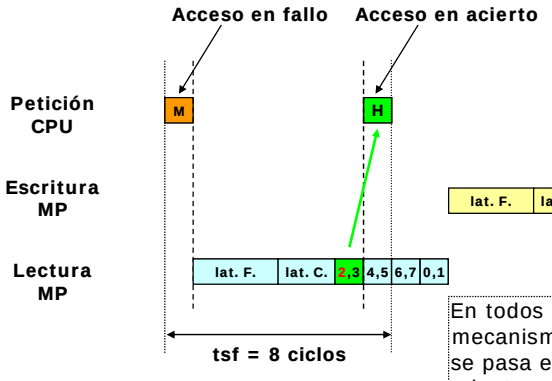
Operaciones a realizar en caso de fallo

- Reducción del tiempo de fallo: Continuación Anticipada (*Early Restart*): en cuanto llega el dato que provoca el fallo, se envía al procesador.



Operaciones a realizar en caso de fallo

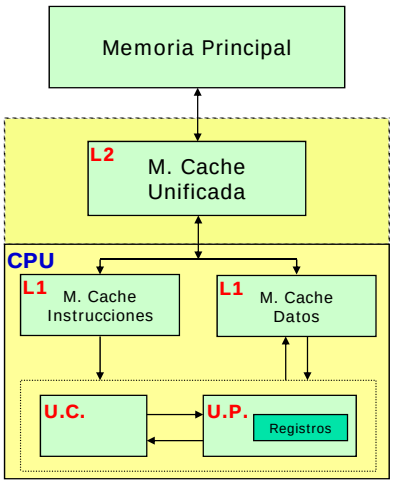
- Reducción del tiempo de fallo: Transferencia en desorden: se envía en primer lugar la palabra que ha provocado el fallo.



En todos los casos, y para que estos mecanismos sean efectivos, cuando se pasa el dato al procesador y mientras se acaba de servir el fallo, la MC ha de ser capaz de recibir nuevos accesos (Cache no bloqueante).

Estructura Básica de un Computador Actual

Problemas: 22-29



Memoria Principal:
- entre 64Mbytes y 4 Gbytes
- Tacceso: entre 40 y 100 ciclos

L2: Memoria Cache Unificada
- entre 128 Kbytes y 4 Mbytes
- Tacceso: entre 10 y 20 ciclos (menor si está integrada)

L1: Memorias Cache Separadas
- entre 8 Kbytes y 64 Kbytes
- Tacceso: entre 1 y 3 ciclos

Registros:
- entre 8 y 128 registros
- bancos separados: INT y FP
- Tacceso: 1 ciclo

Ejemplos de Procesadores

Procesador	Año	L1				L2		
			Asociatividad	Tamaño cache	Tamaño línea	Asociatividad	Tamaño cache	Tamaño línea
SuperSparc II	1995	D	4	16 Kbytes	32 bytes	1	1 Mbyte	32 bytes
		I	5	20 Kbytes	32 bytes			
Pentium Pro	1995	D	2	8 Kbytes	32 bytes	4	256 Kbytes – 1 Mbyte	32 bytes
		I	4	8 Kbytes	32 bytes			
Alpha 21164 *	1996	D	1	8 Kbytes	32 bytes	3	96 Kbytes	32 bytes
		I	1	8 Kbytes	32 bytes			
AMD K6	1997	D	2	32 Kbytes	64 bytes	1	256-512 Kbytes	32 bytes
		I	2	32 Kbytes	64 bytes			
Alpha 21264	1998	D	2	64 Kbytes	64 bytes	1	1-16 Mbytes	64 bytes
		I	2	64 Kbytes	64 bytes			
Pentium III	1998	D	4	16 Kbytes	32 bytes	8	512 Kbytes	32 bytes
		I	4	16 Kbytes	32 bytes			
Itanium I *	2000	D	4	16 Kbytes	32 bytes	6	96 Kbytes	32 bytes
		I	4	16 Kbytes	32 bytes			
Pentium 4 *	2000	D	4	8 Kbytes	64 bytes	8	256 Kbytes	64 bytes
		I	4	8¿? Kbytes	64 bytes			

Los procesadores marcados con un asterisco tienen un tercer nivel de cache. Los datos presentados en esta tabla son datos parciales. De cada procesador se construyen muchos modelos diferentes, variando coste y rendimientos. Es posible que los parámetros de la Memoria Cache varíen notablemente entre un modelo y otro.

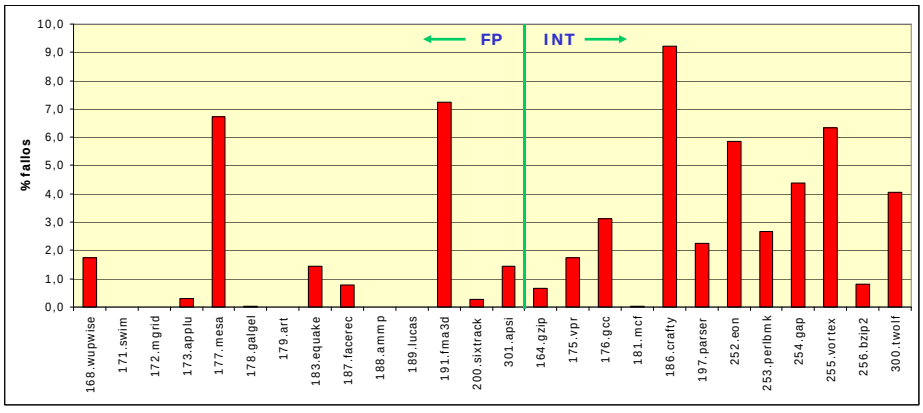
Influencia de los Parámetros de la Cache en el Rendimiento

- Parámetros que influyen en el rendimiento de la Memoria Cache:
  - Tamaño Cache
  - Tamaño de Línea
  - Asociatividad
  - Programas
- Parámetros de rendimiento:
  - Tasa de Fallos (m)
  - Tiempo de ciclo (Tc, tsa)
  - Tiempo medio de acceso (Tma = h·tsa + m·tsf f)
  - Tiempo de ejecución (Texe = N·CPI·Tc)

Influencia de los Parámetros de la Cache en el Rendimiento

- Resultados experimentales obtenidos de la ejecución del Spec 2000:
  - Ejecución completa, datos ref
- Cada ejecución completa equivale $8x10^{12}$ instrucciones simuladas en un procesador Alpha 21264
- Tiempos de ciclo obtenidos a partir de CACTI (¡modificados!).
- Datos:
  - Número de referencias por instrucción (nr) = 0.3624 (¡real!)
  - Ciclos por Instrucción (CPI) = 1.15
  - Coste de leer una línea de MP: 100 ciclos con una memoria de 500 MHz
  - Política de Escritura: COPY BACK + WRITE ALLOCATE

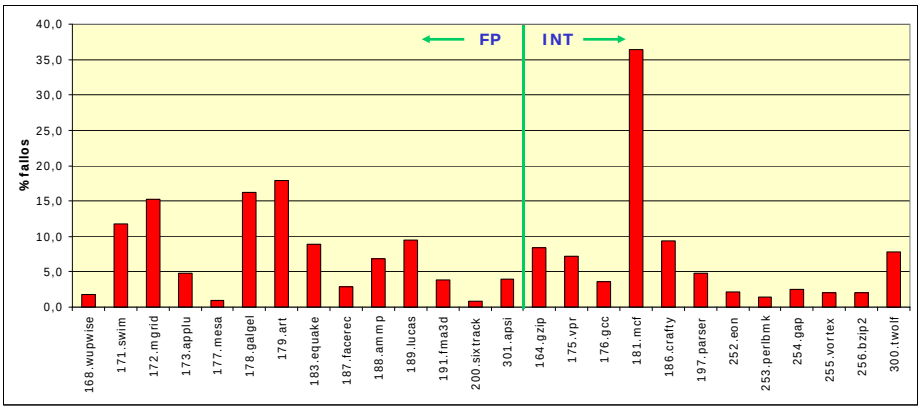
Influencia del programa en la tasa de fallo



Cache de Instrucciones directa de 4 Kbytes y líneas de 32 bytes.

¡El programa influye de forma sustancial en la tasa de fallos!

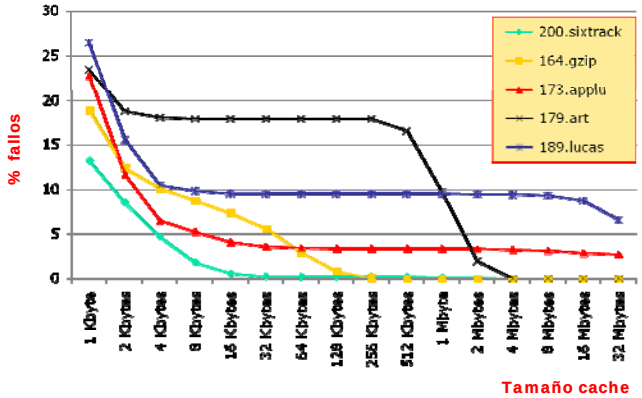
Influencia del programa en la tasa de fallos



Cache de Datos 4-asociativa de 8 Kbytes y líneas de 64 bytes.

¡El programa influye de forma sustancial en la tasa de fallos!

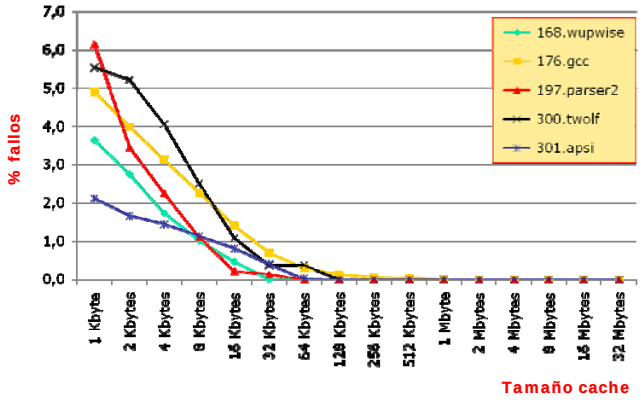
Influencia del tamaño de cache



Cache de datos 2-asociativa con líneas de 64 bytes.

Aumentar el tamaño de cache provoca que la tasa de fallos disminuya.

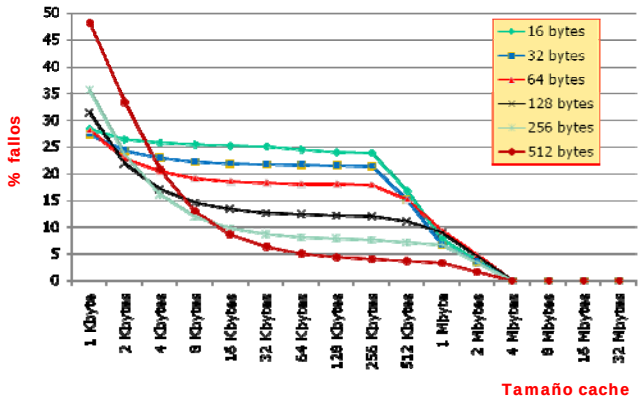
Influencia del tamaño de cache



Cache de instrucciones directa con líneas de 32 bytes.

Aumentar el tamaño de cache provoca que la tasa de fallos disminuya.

Influencia del tamaño de línea

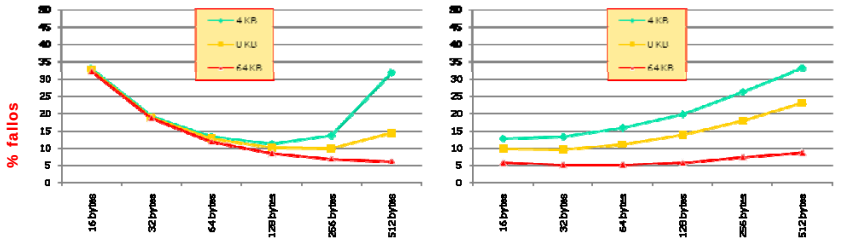


Cache de datos directa.

179.art
programa en C
Cálculo científico
Image Recognition

El tamaño de línea influye, ¡pero, en ambos sentidos!

Influencia del tamaño de línea



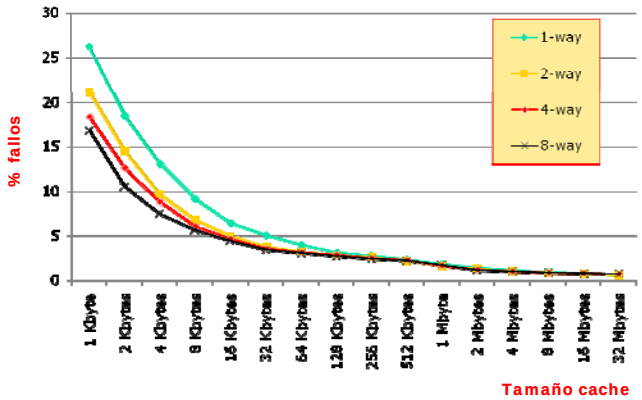
171.swim
programa en fortran 77, cálculo científico
Shallow Water Modeling

175.vpr
Programa en C, enteros
FPGA Circuit Placement and Routing

Cache de datos directa.

El tamaño de línea influye, ¡pero, en ambos sentidos!

Influencia de la Asociatividad

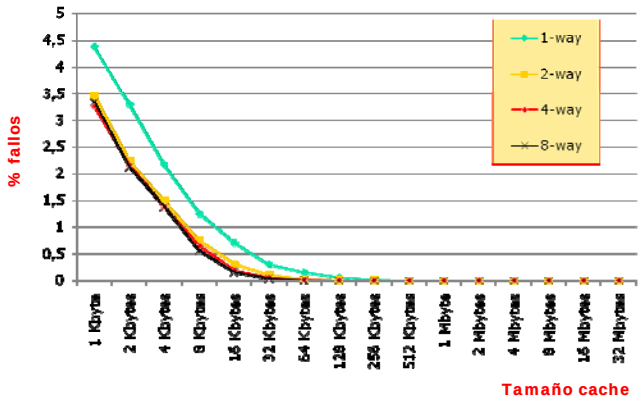


Cache de datos con líneas de 64 bytes.

Aumentar la asociatividad implica disminuir la tasa de fallos

Spec 2000
26 programas
$8 \cdot 10^{12}$ instrucciones ejecutadas
$2{,}9 \cdot 10^{12}$ referencias a memoria

Influencia de la Asociatividad

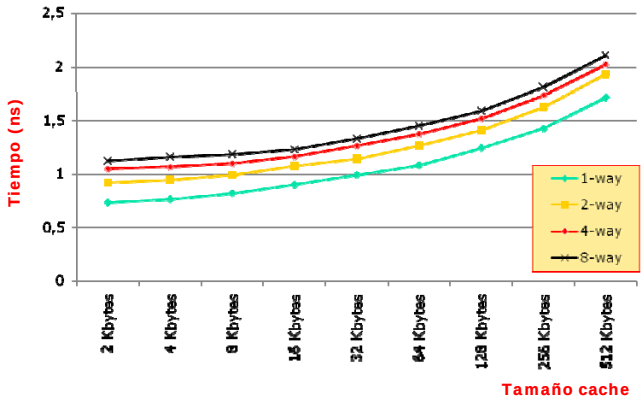


Cache de instrucciones con líneas de 64 bytes.

Spec 2000
26 programas
$8 \cdot 10^{12}$ instrucciones ejecutadas

Aumentar la asociatividad implica disminuir la tasa de fallos

Influencia en el tsa

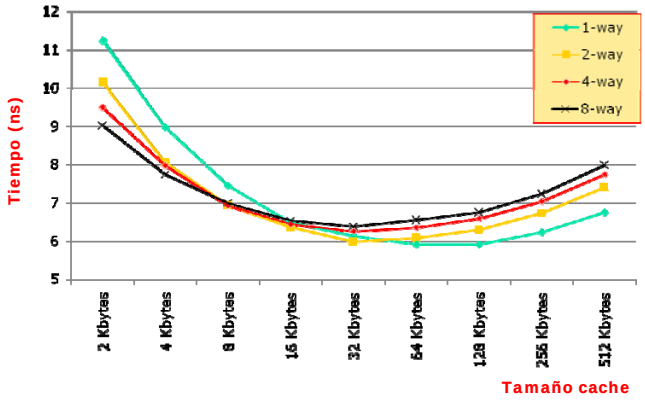


Cache datos con líneas de 32 bytes.

Datos de CACTI (modificado)

El tsa aumenta con la asociatividad y el tamaño de cache

Tiempo medio de acceso (Tma)

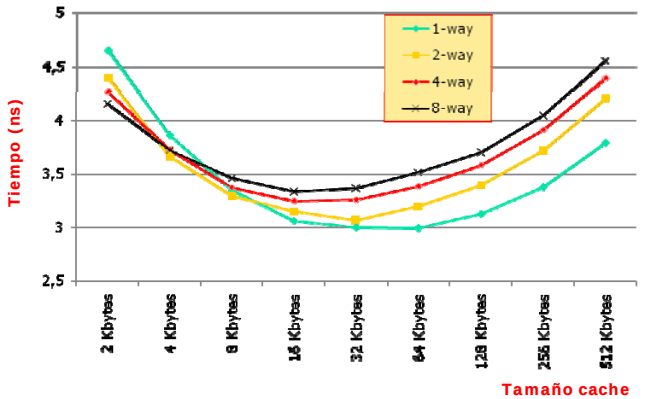


Cache datos con líneas de 32 bytes.

Tma = h·tsa + m·tsf

Spec 2000
Memoria Principal:
- Frecuencia: 500 MHz
- Coste leer línea: 25 ciclos

Tiempo Ejecución de 1 instrucción (Tma)



Cache datos con líneas de 32 bytes.

Texec = 1·(CPIideal + nr·(Tma-tsa))·Tc

Spec 2000
- CPIideal: 1,15
- nr: 0,3624
- Tc: tsa

Mejoras de Rendimiento

- Posibles estrategias:
  - Reducir la tasa de fallos.
  - Reducir el tiempo de penalización por fallo
  - Reducir el coste de las escrituras

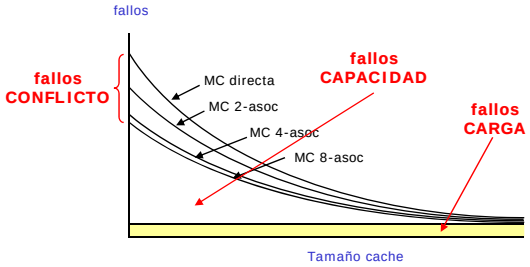
Mejoras de Rendimiento

Los fallos de cache pueden dividirse en tres categorías:

- Carga (compulsory): se producen la primera vez que se accede a una posición de memoria.
- Capacidad: todas las líneas que necesita un programa no caben en la Memoria Cache.
- Conflicto: se producen cuando varias líneas se mapean en el mismo lugar de la MC (sólo en MC directas y asociativas por conjuntos)

¿Cómo se calculan?

¡Ayuda a entenderlo!



Mejoras de Rendimiento: Reducción tasa de fallos

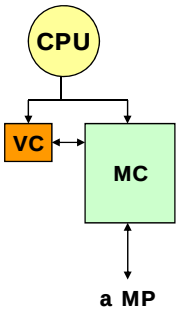
Técnicas Obvias:

- Reducir fallos de carga ⇒ ↑**tamaño de línea** (pueder ser negativo)
- Reducir fallos de capacidad ⇒ ↑**tamaño de cache** (↑tsa)
- Reducir fallos de conflicto ⇒ ↑**asociatividad** (↑tsa y ↑coste)

Mejoras de Rendimiento: Reducción tasa de fallos. Victim Cache

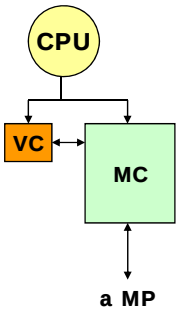
Objetivo: reducir la tasa de fallos manteniendo constante el tsa.

- Idea: añadir una MC completamente asociativa muy pequeña (4-8 líneas) a la MC (normalmente directa).
- Cuando hay un acceso a memoria se accede a las 2 MC, entonces se pueden producir las siguientes situaciones.
  - Acierto en MC
  - Fallo en MC y acierto en VC: se envía el dato a la CPU, además la línea que contiene el dato en la VC se intercambia con una línea de la MC.
  - Fallo en MC y VC, se trae de MP la línea que contiene el dato que ha provocado el fallo y la línea reemplazada se envía a la VC.



Mejoras de Rendimiento: Reducción tasa de fallos. Victim Cache

- Como la VC se puede consultar en paralelo con la MC, el tsa no aumenta.
- Aunque la VC es completamente asociativa, el tsa puede ser similar al de la MC (directa) porque es muy pequeña.
- La VC reduce los fallos debidos a CONFLICTOS.
- Una VC de 4 líneas, puede eliminar entre un 20-95% (dependiendo del programa) de los fallos de conflicto en una MC directa de 4Kbytes.



Mejoras de Rendimiento: Reducción tasa de fallos. PreFetch

Objetivo: Reducir los fallos de CARGA

Estrategia: traer a MC la información antes de que sea solicitada.

Prefetch hardware (datos e instrucciones)

- Idea: cuando se accede a la línea i, se trae la línea i+1 (si no está en MC).
- Problema: Se pueden traer datos inútiles.
- Se puede implementar con un buffer de prefetch (Alpha 21064)
  - Cuando hay un fallo sobre la línea i
    - Se trae la línea i a la MC y la línea i+1 al buffer
  - Cuando hay un fallo sobre la línea i+1
    - Se sirve el fallo desde el buffer (funciona en paralelo con la MC), se pasa la línea i+1 a la MC y se trae la línea i+2 al buffer.
  - El buffer puede tener varias entradas.

Mejoras de Rendimiento: Reducción tasa de fallos. PreFetch

Prefetch software (sólo datos)

- Se utilizan instrucciones especiales (las insertan el compilador o bien el programador de LM) para traer los datos de forma anticipada.
- En general, el prefetch puede introducir tráfico (entre MP y MC) innecesario. Con el prefetch software tenemos más control (se puede reducir el tráfico inútil).
- Se pierde tiempo ejecutando instrucciones de prefetch (¿inútiles?).

Mejoras de Rendimiento : Reducir tiempo de penalización por fallo

- Continuación anticipada (*early restart*)
- Enviar la palabra crítica en primer lugar (transferencia en desorden)
- Caches multinivel
- Non-Blocking caches:
  - El procesador no se para cuando se produce un fallo en MC.
  - El procesador sólo se para cuando la instrucción a ejecutar no tiene los operandos que necesita.
  - Procesadores con ejecución fuera de orden.

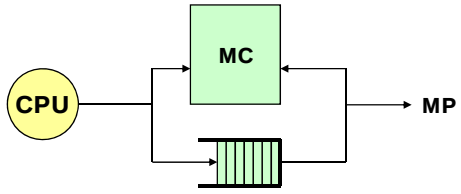
Mejoras de Rendimiento : Reducir el coste de las escrituras

- Cuando se realiza una lectura, se puede leer el dato en paralelo con la comprobación de si es un acierto o un fallo.
- Una escritura en MC no se puede realizar hasta que sabemos que es un acierto.
- Si una lectura nos cuesta 1 ciclo ⇒ una escritura cuesta 2 ciclos.
- Solución: ESCRITURAS SEGMENTADAS
  - En el primer ciclo se comprueba si es acierto o fallo y en caso de acierto se deja el dato a escribir en un registro intermedio.
  - En la siguiente escritura (mientras se comprueba si es acierto o fallo) se realiza la escritura anterior.
  - Una escritura individual sigue tardando 2 ciclos, pero desde el punto de vista del procesador sólo cuesta un ciclo.

Mejoras de Rendimiento : Reducir el coste de las escrituras

Buffers de Escritura

- Si la cache es WRITE THROUGH, las escrituras se realizan a la velocidad de MP (no es aceptable).
- Solución: poner un buffer de n entradas entre la CPU y MP.
  - El buffer es rápido (equivale a acceder a MC, 1 ciclo).
  - Las escrituras se realizan cuando el bus entre MC y MP está libre.
  - Unas pocas entradas son suficientes.



•Cuando se accede a MC, también hay que consultar el buffer.
• Se puede utilizar un buffer similar para una MC con COPY BACK.

Aspectos de programación

- La Memoria Cache es transparente al programador. Sin embargo, tenerla en cuenta en la programación puede mejorar (a veces espectacularmente) el rendimiento de los programas.
- Existen técnicas de programación / compilación que permiten optimizar el rendimiento de la jerarquía de memoria.

Fusión de Arrays

- Situación original:

```
int val[N];
int key[N];
for (i=0; i<N; i++) {
  … val[i]…
  … key[i]…
}
```

Problema:

- Esta situación puede provocar conflictos graves si **val[i]** y **key[i]** se mapean en la misma línea de la cache.
- Se desaprovecha localidad espacial. Todos los accesos serán fallos.

Fusión de Arrays

- Solución

```
struc record {
  int val;
  int key;
};
```

```
struct record fusionado[N];
for (i=0; i<N; i++) {
  … fusionado[i].val…
  … fusionado[i].key…
}
```

- Se evitan los conflictos
- Además, se aprovecha la localidad espacial. **`fusionado.val[i]`** y **`fusionado.key[i]`** están en la misma línea de cache o en líneas consecutivas

Padding

- Situación original

```
int A[N], B[N];
for (i=0; i<N; i++){
  … A[i]…
  … B[i] …
}
```

Problema:

- Posibilidad de conflicto si **`A[i]`** y **`B[i]`** se mapean en la misma línea de cache (mismo caso que el anterior)
- No se aprovecha localidad espacial.

Padding

- Solución

```
int A[N], dummy[cte], B[N];
for (i=0; i<N; i++)
  A[i]… B[i]
```

¿Tamaño mínimo de la constante?

- Desalinear las direcciones iniciales de **A[i]** y **B[i].**
- Se aprovecha localidad espacial.
- El mismo problema aparece si se define una matriz

  int A[1024][1024];

  y se accede por columnas.
- Solución:

  int A[1024][1024+k];

Intercambio de bucles

- Situación original

```
int A[M][N];
for (i=0; i<N; i++)
  for (j=0; j<M; j++) {
  … A[j][i]…
  }
```

Problema:

- Accedemos a datos distanciados N·4 bytes
- No hay localidad espacial
- Además, si N·4 es múltiplo del tamaño de la cache, todos los accesos provocarán conflictos.

Intercambio de bucles

- Solución: intercambio de bucles

```
int A[M][N];
for (j=0; j<M; j++)
  for (i=0; i<N; i++) {
  … A[j][i]…
  }
```

- Accedemos a posiciones consecutivas (matriz almacenada por filas).
- Aprovechamos la localidad espacial.

¡ATENCIÓN! El intercambio de bucles no siembre es legal. Hay que mantener la semántica del programa.

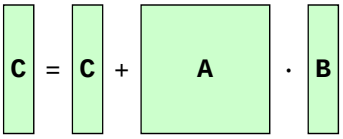
Blocking

- Situación original

```
int A[N][N];
int B[N], C[N];
for (i=0; i<N; i++)
  for (j=0; j<N; j++) {
  C[i]=C[i]+A[i][j]*B[j]
  }
```

Problema

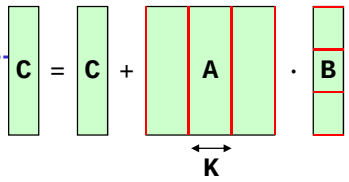
- La matriz se recorre por filas (localidad espacial)
- Pero, si tamaño B > tamaño cache, no se aprovecha localidad temporal



Blocking

- Solución

```
for (jj=0; jj<N; jj=jj+K)
  for (i=0; i<N; i++)
    for (j=jj; j<min(N,jj+K); j++)
      C[i]=C[i]+A[i][j]*B[j]
```



- K se escoge de tal forma que la porción del vector B que se utiliza en cada iteración de jj quepa en la MC: aprovechamos la localidad temporal.