UT 3. Subsistema de memoria

Tema 3.2 Mejora de las prestaciones de las memorias cache

A. Doménech, J. Duato, P. López, V. Lorente,
A. Pérez, S. Petit, J.C. Ruiz, S. Sáez, J. Sahuquillo

Departamento de Informática de Sistemas y Computadores
Universitat Politècnica de València

Índice

Bibliografía

John L. Hennessy and David A. Patterson.
Computer Architecture, Fifth Edition: A Quantitative Approach.
Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 5 edition, 2012.

Índice

Tiempo de acceso medio

$T_{\text{acceso}} = TA + TF \times PF$

donde TA: tiempo de acierto en cache
TF: tasa de fallos
PF: penalización de fallo

¿Cómo mejorar las prestaciones de las cache?
Reduciendo cualquiera de los términos:

- Penalización por fallo (*PF*).
- Tasa de fallos (*TF*).
- Tiempo en caso de acierto (*TA*).

Índice

Técnicas

- Caches multinivel.
- *“Critical word first”* y *“Early restart”*.
- Buffers de escritura combinadas.

Caches multinivel

- Nuevo nivel de cache (L2) ubicado entre la memoria cache y la memoria principal:
  - La cache L1 es lo suficientemente pequeña como para ser tan rápida como el procesador.
  - La cache L2 es lo suficientemente grande como para capturar muchos de los accesos a memoria principal.
- ¿Cómo cambia la ecuación del tiempo de acceso?

$$T_{\text{acceso}} = TA_{L1} + TF_{L1} \times PF_{L1}$$

La penalización por fallo de cache L1 es:

$$PF_{L1} = TA_{L2} + TF_{L2} \times PF_{L2}$$

Caches multinivel (cont.)

Sustituyendo:

$$T_{\text{acceso}} = TA_{L1} + TF_{L1} \times (TA_{L2} + TF_{L2} \times PF_{L2})$$

Caches multinivel (cont.)

- Con varios niveles de cache distinguimos dos tipos de tasas de fallos:

  Tasa de fallos local= $\frac{\text{Num. de fallos de la cache}}{\text{Num. total accesos cache}}$

  - Para 2 niveles tendremos: TF_{L1} y TF_{L2}

  Tasa de fallos global= $\frac{\text{Num. de fallos de la cache}}{\text{Num. total accesos}}$

  - Para la cache L1 es TF_{L1}
  - Para la cache L2 es

    $$\frac{\text{Fallos L2}}{\text{Total accesos}} = \frac{\text{Fallos L1}}{\text{Total accesos}} \cdot \frac{\text{Fallos L2}}{\text{Fallos L1}} = \frac{\text{Fallos L1}}{\text{AccesosL1}} \cdot \frac{\text{Fallos L2}}{\text{Accesos L2}} = TF_{L1} \times TF_{L2}$$

  $\rightarrow$ TF $Global_{L2} = TF_{L1} \cdot TF_{L2}$ es la fracción de accesos que llegan a memoria.

- Desde el punto de vista de la tasa de fallos:
  - La tasa de fallos global de un sistema con dos niveles de cache es similar a la que obtendríamos con un sistema con un sólo nivel del mismo tamaño que la L2.
  - ...pero con una cache L1 de menor tamaño $\rightarrow$ más rápida.

Caches multinivel (cont.)

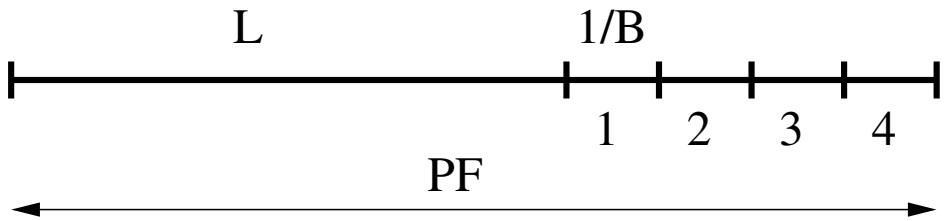
- Algunos aspectos de diseño:
  - La velocidad de la cache L1 afecta al ciclo de reloj del procesador
    ¿Cómo reducir TA_{L1} (ver página 42)
    - Cache L1 pequeña y con correspondencia directa o asociativa de pocas vías.
  - La cache L2 afecta la penalización por fallo de L1
    ¿Cómo reducir $PF_{L1} = TA_{L2} + TF_{L2} \times PF_{L2}$?
    El segundo término es el dominante al estar multiplicando $PF_{L2} \rightarrow$
    Reducir TF_{L2} (ver página 21):
    - Cache L2 grande, mucho mayor que L1, y con bastantes más vías.

Caches multinivel (cont.)

- Inclusión/exclusión multinivel
  - Inclusión multinivel
    - Los datos que están en la cache L1 están siempre en la cache L2.
    - Ayuda a mantener la coherencia. Sólo es necesario comprobar el nivel inferior (L2).
    - En algunos procesadores el tamaño de bloque de L2 es superior al de L1, o el de L3 al de L2. En caso de reemplazamiento en el nivel inferior, ej L2, hay que invalidar todos los bloques del superior (L1) que componen el bloque de L2, aumentando TF_{L1}.
  - Exclusión multinivel (AMD Opteron)
    - No hay bloques replicados en las caches: están en L1 o en L2 $\rightarrow$ mejor aprovechamiento del espacio.
    - Un fallo en L1 y acierto en L2 intercambia bloques entre las caches.
    - Interesante cuando el tamaño de la cache L2 es sólo ligeramente superior al de la cache L1.
    - Dificulta la implementación de mecanismos de coherencia.

"Critical word first" y *"Early restart"*

La penalización por fallo *PF* depende de la latencia *L* y ancho de banda *B* de la memoria:



$PF = L + \frac{1}{B} \cdot n$, siendo n el tamaño de bloque.

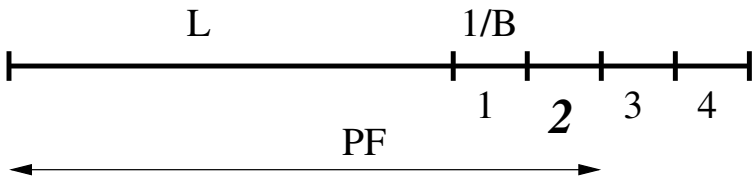
- Latencia *L* o tiempo de acceso. Desde que se envía la dirección hasta que el primer dato está disponible a la salida de la memoria.
- Ancho de banda *B*. Velocidad con la que pueden suministrarse los datos.

"Critical word first" y *"Early restart"* (cont.)

El procesador sólo necesita la palabra del bloque que ha provocado el fallo. → ¿Porqué esperar a tener todo el bloque cargado para entregar la palabra?

Soluciones:

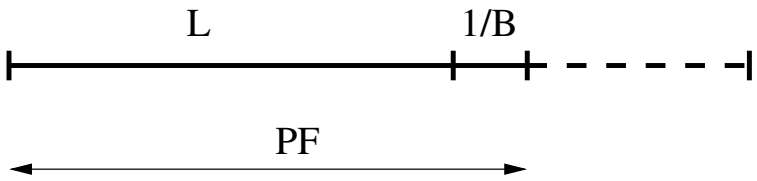
Early restart:



- Acceder a las palabras del bloque normalmente.
- En cuanto llega la palabra solicitada se le entrega al procesador para que continue con la ejecución.

"Critical word first" y *"Early restart"* (cont.)

Critical word first:



- Accede primero en memoria a la palabra solicitada por el procesador.
- Trae el resto del bloque mientras el procesador continua con la ejecución.

Las mejoras obtenidas por *Critical word first* y *Early restart* son tanto mayores cuando:

- Se emplean tamaños de bloque grandes.
- El siguiente acceso a memoria[1] *no referencie* el mismo bloque que se está cargando $\rightarrow$ en ese caso, el segundo acceso debe esperarse a que el bloque esté cargado.

[1] Suponiendo cache no bloqueante

Buffers de escritura

Problema de las escrituras: la memoria es mucho más lenta que el procesador $\rightarrow$ las escrituras pueden parar al procesador.

- ¿Cuando se escribe en memoria?

  Write-through: Siempre que se escribe en la cache.
  Write-back: Cuando se reemplaza un bloque ”sucio”.
- Las escrituras no son instrucciones “productoras” $\rightarrow$ no deben parar al procesador.

Solución: buffer de escritura

- El procesador escribe sobre el buffer sin parar la ejecución, desacoplando la ejecución de la escritura en memoria, que la lleva a cargo el controlador.

Buffers de escritura (cont.)

- Problema: dependencias de datos con la memoria.

```
1 sd r3,a(r10)  ;  Escritura con fallo en Mem[a+r10]
                ;  Write-through y No-Write Allocate
2 ld r1,b(r11)  ;
3 ld r2,a(r10)  ;  Lectura con fallo de Mem[a+r10]
```

  $\rightarrow$ Si la escritura del r3 (instr 1) no se ha realizado cuando se lee el r2 (instr 3), el valor cargado es incorrecto

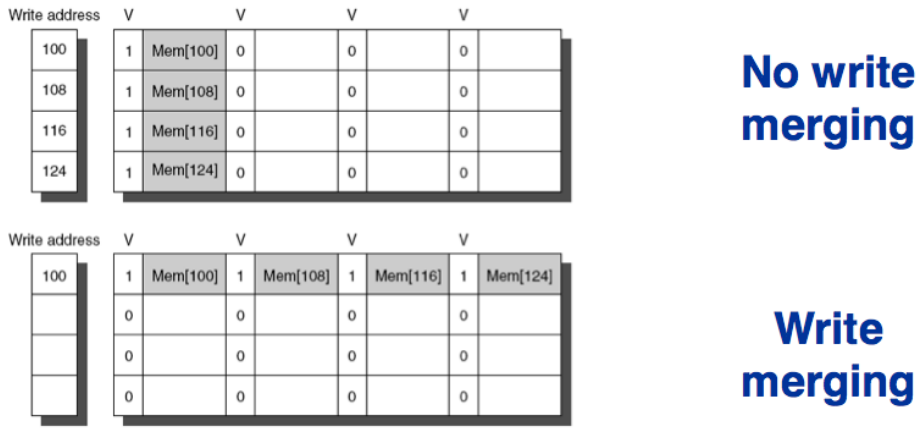
- Soluciones:
  - Esperar a que el buffer de escritura se vacíe antes de leer el dato.
  - Comprobar si la dirección referenciada está en el buffer de escritura y, si no está, dejar que la lectura continúe (*load-bypassing*).
  - Si la dirección referenciada está en el buffer de escritura, leer el dato del buffer (*load-forwarding*).

Buffer de escrituras combinadas

Si el write buffer está lleno y se necesita una entrada $\rightarrow$ *stalls*.

Solución: buffers de escrituras combinadas

- Cada entrada del buffer hace referencia a un conjunto de direcciones consecutivas.
- Se compara la dirección con la de las entradas válidas del buffer. Si coincide, el dato se combina con dicha entrada.



Las entradas de la derecha de la figura superior sólo la utilizan instrucciones multipalabra.

Buffer de escrituras combinadas (cont.)

- Uso eficiente de la memoria: se escriben varias palabras a la vez en vez de una en una.
- El Sun T1 (Niagara) y el Intel Core i7, entre muchos otros, utilizan escrituras combinadas para conseguir un *Write-through* rápido L1 a L2.

Índice

Clasificación de los fallos de bloque

Arranque (*compulsory* Originados la primera vez que se accede a un bloque.
→ Suelen representar un bajo porcentaje del total de fallos.

Conflicto Aparecen cuando el conjunto destino está lleno pero hay espacio en otros conjuntos.
→ No hay en una cache totalmente asociativa, pero necesita mucho hardware y puede reducir la frecuencia de reloj.

Capacidad Si la cache no puede alojar todos los bloques necesarios durante la ejecución de un programa, hay bloques activos que se reemplazan, por lo que se producirán fallos de capacidad.
→ Se reducen al incrementar el tamaño de la cache.

La técnicas para reducir la tasa de fallos pretenden reducir la tasa global.

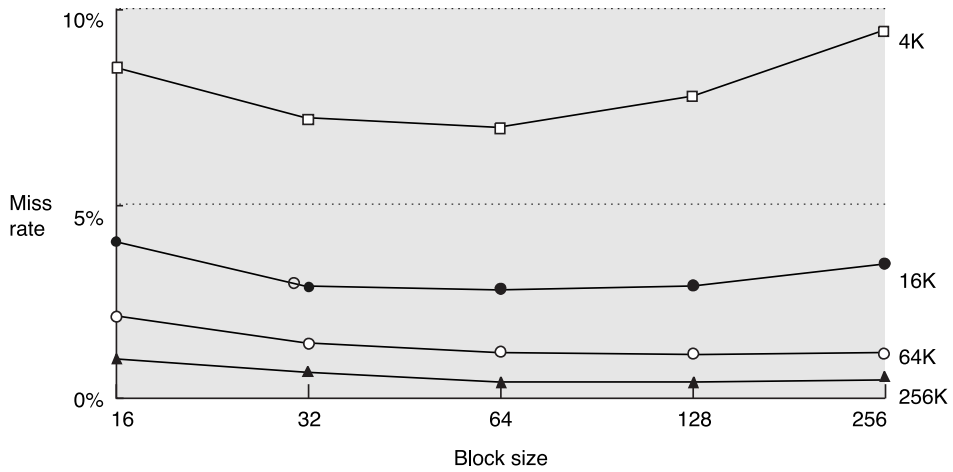
Técnicas

- Ajustar la geometría:
  - Tamaño de bloque.
  - Tamaño de la cache.
  - Número de vías.
- Predicción de vía.
- Optimizaciones del compilador.

Tamaño de bloque

Un tamaño de bloque más grande ...

- Explota más la localidad espacial $\rightarrow$ reduce fallos de arranque($\downarrow$ *TF*).
- Reduce el número de líneas de cache para una capacidad dada.
  - Puede empeorar los fallos por conflicto ($\uparrow$ *TF*).
  - Puede empeorar los fallos por capacidad ($\uparrow$ *TF*).



Tamaño de bloque (cont.)

- Aumentar el tamaño de bloque aumenta la penalización en caso de fallo ($\uparrow PF$), ya que hay que traer más palabras.
- ¿Cuál debe ser el tamaño del bloque?
  Recuerda $T_{\text{acc}} = TA + TF \times PF$
  - El tamaño de bloque es un compromiso entre TF y PF.
    Debe ser relativamente grande para amortizar PF, pero la tasa de fallos debe ser razonable.
- El tamaño típico es 64B.
  Algunos procesadores ponen un tamaño más grande en el último nivel de cache (LLC, *Last Level Cache*).
  Ejemplo: en el IBM power 7 el tamaño de bloque de L3 es 4 veces el de L2.

Tamaño de la cache

Un tamaño de cache más grande ...

- Reduce los fallos por capacidad ($\downarrow$ *TF*).
- Aumenta el tiempo en caso de acierto ($\uparrow$ *TA*).
- Aumenta el coste en área y consumo.
- Idea: diseñar L1 pequeña y rápida para prestaciones y
  L2 grande para capacidad y
  usar tecnologías de bajo consumo (v.g. eDRAM) para caches de gran capacidad.

El tamaño de L1 típico suele ser de 16KB a 32KB.
Las LLC suelen ser muy grandes (decenas de MB) para proporcionar un buen compromiso entre TF y PF.

Número de vías

Un mayor número de vías ...

- Reduce los fallos por conflicto ($\downarrow$ *TF*).
- Requiere más comparadores (más consumo de energía).
- El multiplexor de vía tiene más entradas
  $\rightarrow$ puede aumentar el tiempo en caso de acierto ($\uparrow$ *TA*)

Las caches de L1 suelen tener de 2 a 8 vías, mientras que las LLC superan a veces las 20 vías.

Predicción de vía

Objetivos:

- Reducir los fallos por conflicto sin aumentar TA respecto a una de correspondencia directa.
- Reducir energía respecto a una asociativa.

Idea: predicción de vía (Alpha 21264):

- La cache es asociativa por conjuntos.
- La cache incluye un predictor de qué bloque del conjunto se referenciará la próxima vez que se acceda a la cache.
  $\rightarrow$ Se pre-configura el multiplexor de vía accediendo en paralelo a la comprobación de etiqueta y al dato.
- Si la predicción es incorrecta, se comparan el resto de etiquetas.

Predicción de vía (cont.)

- Hay dos tiempos en caso de acierto:
  - **Predicción OK:** solo se compara una etiqueta $\rightarrow$ TA bajo (1 ciclo).
  - **Fallo de predicción:** se comparan el resto de etiquetas, se actualiza el predictor y se configura el multiplexor de vía. $\rightarrow$ TA más elevado (3 ciclos).

El predictor acierta en el 85 % de los casos.
Su comportamiento es similar a una cache de dos niveles.

$$T_{\text{acceso}} = \%\ \textit{aciertos} \times TA_{\text{acierto pred.}} + \%\ \textit{fallos} \times TA_{\text{fallo pred.}}$$

En el ejemplo:

$$T_{\text{acceso}} = 0{,}85 \times 1 + 0{,}15 \times 3$$

Optimizaciones del compilador

El compilador genera un código optimizado que reduce la tasa de fallos.

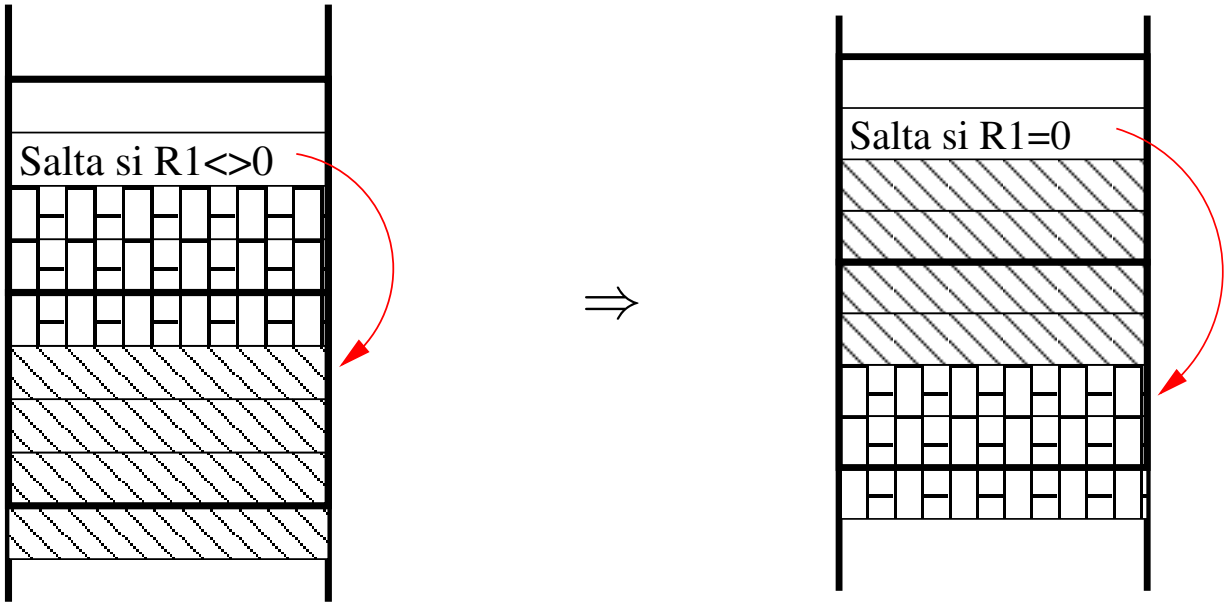
Reducción de la tasa de fallos de instrucciones:

- Reordenación de los procedimientos de un programa de forma que se ubiquen en conjuntos distintos en la cache para reducir los fallos por conflicto.
- Alinear el punto de entrada de los bloques básicos con el principio de un bloque de cache $\rightarrow$ mejora la localidad espacial.

Optimizaciones del compilador (cont.)

- *Branch straightening*. Si el compilador cree que un salto será "tomado", se modifica el código para:
  - evaluar la condición contraria, y
  - ubicar a continuación de la instrucción de salto el código antes ubicado en el destino del salto

$\rightarrow$ mejora la localidad espacial.



Optimizaciones del compilador (cont.)

Reducción de los fallos de datos:

Ejemplo: operaciones con matrices. Reorganizar el código para operar sobre todos los datos de un bloque antes de pasar al siguiente.

Mejorar la localidad espacial Intercambio de bucles.

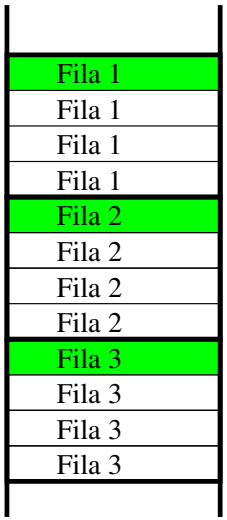
Cambiando el anidamiento de los bucles podemos conseguir operar en el orden en que están almacenados los datos.

Optimizaciones del compilador (cont.)

Ejemplo, matriz x almacenado por filas:

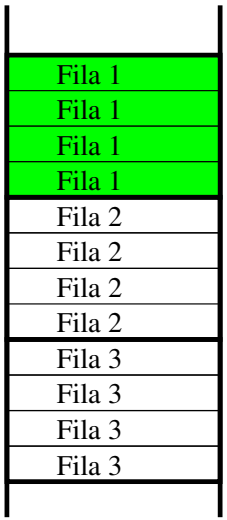
```
/* antes */
for (j=0; j<100; j=j+1)
   for (i=0; i<5000; i=i+1)
      x[i][j] = 2* x[i][j]
```

→ salta 100 palabras por acceso.




```
/* despues */
for (i=0; i<5000; i=i+1)
   for (j=0; j<100; j=j+1)
      x[i][j] = 2* x[i][j]
```

→ accede secuencialmente.



Optimizaciones del compilador (cont.)

Mejorar la localidad temporal: *blocking*

Si se accede tanto por filas como por columnas, es mejor operar sobre submatrices o bloques, para maximizar los accesos a los datos cargados en la cache antes de reemplazarlos.

Ejemplo: multiplicación de matrices

```
/* 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;
  }
```

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

Índice

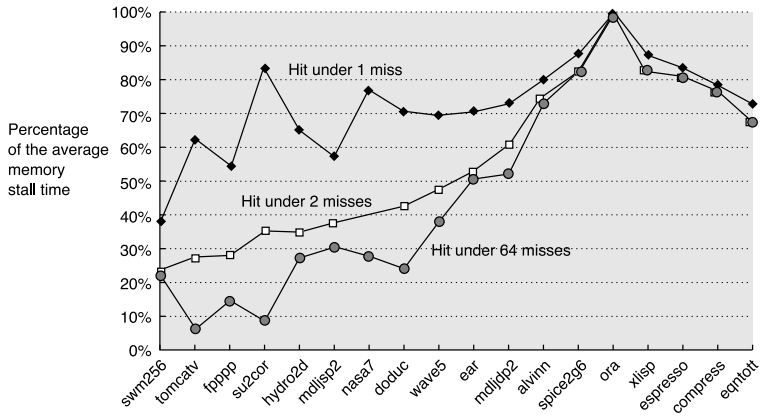
Concepto y técnicas

$\rightarrow$ técnicas que solapan la ejecución de instrucciones en el procesador con el acceso a la memoria.

- *Cache* no bloqueante
- Pre-búsqueda *hardware* de instrucciones y datos
- Pre-búsqueda controlada por el compilador

Cache no bloqueante

- La cache continua aceptando peticiones de acceso mientras se está sirviendo un fallo de cache.
- Posibilidades:
  - "acierto ante fallo": la cache solo puede gestionar un fallo, pero sirve nuevas peticiones si son aciertos.
  - "fallo ante fallo"/"acierto ante múltiples fallos": se pueden servir múltiples fallos simultáneamente.



Pre-búsqueda *hardware* de instrucciones y datos

Idea: buscar la información antes de que sea solicitada por el procesador.

- La información pre-búscada se almacena en un buffer externo, que se accede más rápido que la memoria principal.
- Pre-búsqueda de instrucciones.
  - El procesador trae dos bloques ante un fallo: el bloque solicitado, que se ubica en la cache y el consecutivo, que se ubica en el buffer de instrucciones.
  - Cuando se produce un fallo de cache, se lee el bloque del buffer y si está disponible y prebusca el siguiente bloque.
  - Evaluación de la propuesta.
    Cache: 4KB, bloques de 16B.
    Fallos evitados: 1 buffer evita de 15 %–25 %, 4 buffers un 50 % y 16 buffers un 72 %.

Pre-búsqueda *hardware* de instrucciones y datos (cont.)

- Pre-búsqueda de datos.
  - Más compleja que la de instrucciones.
  - La lógica del prefetcher calcula los bloques a prebuscar. La mayoría de los prefetchers actuales detectar patrones de stride regulares no unitarios.
- Las peticiones de prebusca compiten con los fallos de cache en el acceso a memoria principal $\rightarrow$ puede aumentar la penalización por fallo.

Pre-búsqueda controlada por el compilador

- El compilador inserta instrucciones de “pre-búsqueda” para solicitar los datos antes de que se necesiten.
- Las instrucciones de pre-búsqueda no deben generar fallos de página ni excepciones por violación de protección → *nonbinding fetch*
- Tiene sentido con caches no bloqueantes.
- Especialmente útil en bucles. Ejemplo:
  Cache 8KB, bloques de 16B, correspondencia directa. Write-back, write allocate.
  a es una matriz 3x100 y b es una matriz 101x3, ambas de números en coma flotante (8B).
  Supongamos que la PF es tan elevada que pre-buscamos 8 iteraciones
  No prebuscamos los primeros accesos, ni eliminamos las últimas pre-búsquedas.

```
/* original */
for (i=0; i<3; i=i+1)
  for (j=0; j<100; j=j+1)
      a[i][j] = b[j][0]*
                b[j+1][0];
```

Total fallos: 150(a)+101(b)=251

```
/* con pre-búsqueda */
for (j=0; j<100; j=j+1) {
/* b[j,0] para 8 it. después */
  prefetch(b[j+8][0])
/* a[0,j] para 8 it. después */
  prefetch(a[0][j+8])
  a[0][j] = b[j][0]*
            b[j+1][0];};
for (i=1; i<3; i=i+1)
  for (j=0; j<100; j=j+1) {
/* a[i,j] para 8 it. después */
      prefetch(a[i][j+8])
      a[i][j] = b[j][0]*
                b[j+1][0];};
```

Total fallos: 8 (b) + 4x3 (a) = 20

Instrucciones extra: 400

- Sobrecarga: las instrucciones de pre-búsqueda aumenta el número de instrucciones ejecutadas $\rightarrow$ hay que concentrarse en los accesos que serán fallos de bloque con una probabilidad alta.

Índice

Concepto y técnicas

Reducir el tiempo en caso de acierto es muy importante: afecta al periodo de reloj del procesador (la T de la ecuación del tiempo de ejecución).

- Caches pequeñas y sencillas
- Evitar la traducción de MV durante el acceso a la cache
- Segmentación de la cache

Caches pequeñas y sencillas

- Un gran % del tiempo de acceso a la cache se invierte en comparar el campo de etiqueta de la dirección con las etiquetas almacenadas en la cache.
- Modo de reducir este tiempo:

  Cache pequeña: "el hardware pequeño es más rápido" y puede caber en el mismo chip que el procesador.

  Cache sencilla: correspondencia directa (no usan el multiplexor de vía) o pocas vías.

  - Correspondencia directa = 1.2 - 1.5 más rápida que asociativa de 2 vías.
  - Asociativa de 2 vías = 1.02 - 1.11 más rápida que asociativa de 4 vías.
  - Asociativa de 4 vías = 1 - 1.08 más rápida que asociativa de 8 vías.

Caches pequeñas y sencillas (cont.)

- Tendencia:

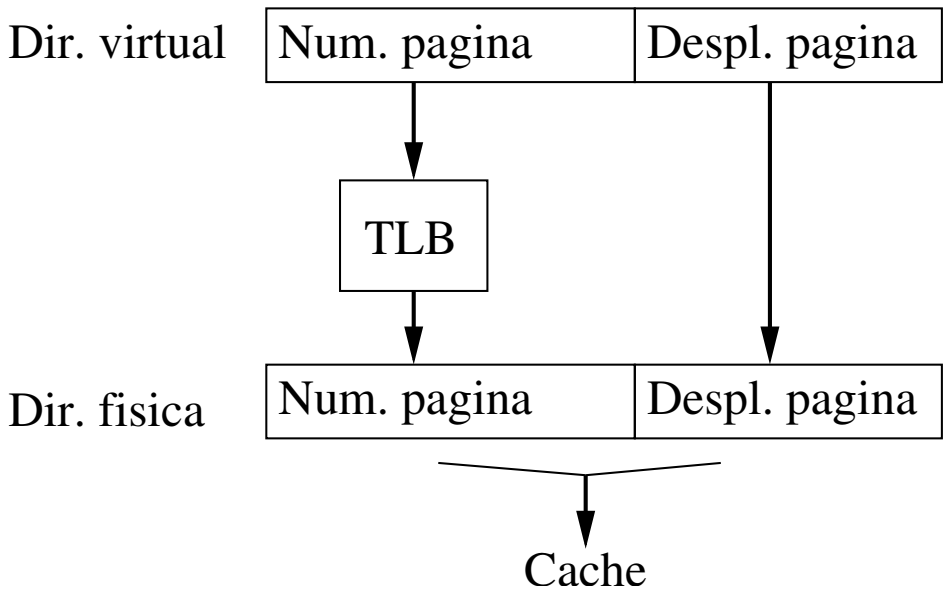
  Caches L1: Énfasis velocidad similar al procesador.
  - Tamaño reducido y correspondencia asociativa de pocas vías.
  - Muchas L1 implementan 2 vías aunque algunos cores tienen 8.

  Cache L2: Mayor tamaño con el fin de evitar los accesos a memoria.
  - Suelen ser privadas (accedidas por un solo core) o compartidas por varios cores.

  Cache L3: algunos procesadores incluyen L3 como LLC.
  - Estas caches son compartidas por varias L2.
  - Se diseñan para evitar más aún los accesos a memoria.
  - Son enormes y suelen tener más de 20 vías.
  - Se utilizan tecnologías de bajo consumo para su implementación.

Evitar la traducción de MV durante el acceso a la cache

Otra componente del tiempo en caso de acierto es el invertido en traducir la dirección virtual emitida por el procesador en una dirección física de memoria.
Hasta que no se realiza la traducción no se puede acceder a la cache.



Evitar la traducción de MV durante el acceso a la cache (cont.)

- Idea. Utilizar direcciones virtuales en la cache.
  Caches virtuales vs. caches físicas
- Varios problemas:

  Protección Es parte del proceso de traducción dirección virtual a física
  $\rightarrow$ Hay que copiar información de la TLB a la cache.

  Procesos Cada proceso tiene su propio espacio de direcciones virtuales. Cada vez que se cambia de contexto, una misma dirección virtual apunta a una dirección física distinta.
  $\rightarrow$ Hay que vaciar la cache con cada cambio de contexto o bien añadir identificadores de proceso (PID) a las etiquetas de la cache.

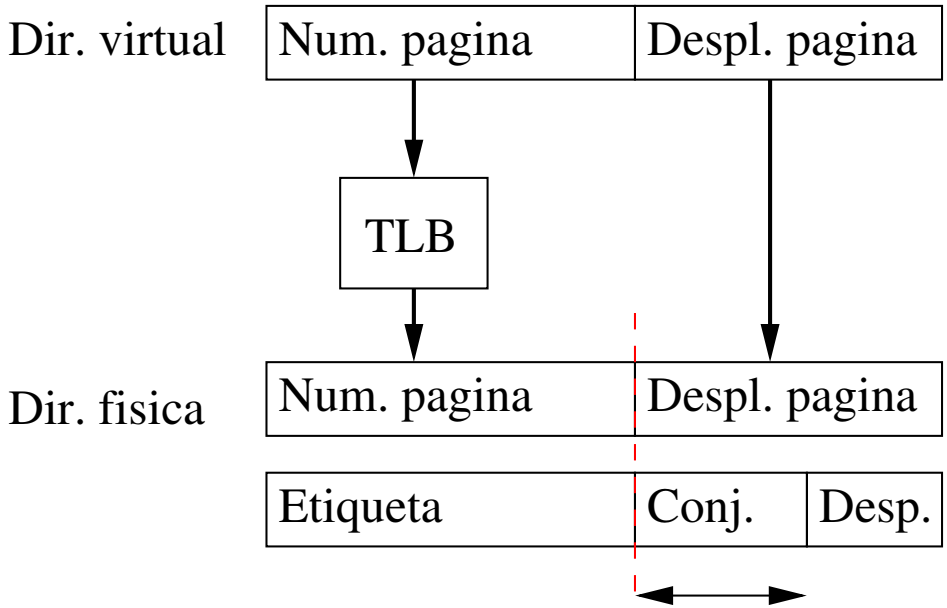
Evitar la traducción de MV durante el acceso a la cache (cont.)

Sinónimos o alias Una misma dirección física puede referenciarse mediante dos o más direcciones virtuales.
$\rightarrow$ Hay varias copias del mismo dato, que deben mantenerse idénticas.

- *Virtually indexed physically tagged caches* (Alpha 21264).

Observación: la dirección dentro de la página (*page offset*) es la misma, tanto en la dirección virtual como en la física.

Evitar la traducción de MV durante el acceso a la cache (cont.)



- Utiliza parte del desplazamiento de página para indexar el conjunto (campo de “Índice”).
- La lectura de la cache se realiza en paralelo con la traducción.
- Limitación:
  El tamaño de una cache con correspondencia directa o el número de conjuntos de una cache con correspondencia asociativa no puede exceder el tamaño de página de memoria virtual.

Segmentación de la cache

- Segmenta el acceso a la cache para alcanzar la frecuencia de reloj del procesador.
- Un acceso a la cache requiere varios ciclos de reloj (por ejemplo, 4 ciclos en el Pentium 4) pero varias instrucciones pueden solapar su acceso.
- Tiene implicaciones en la segmentación del procesador: más ciclos de parada en los saltos y en las operaciones de carga.

Esta técnica incrementa el ancho de banda de instrucciones que acceden a la cache más que reducir la latencia.