ARQUITECTURA E INGENIERÍA DE COMPUTADORES

Tema 3.1

Miquel Gómez Corral

Ing. Informática ETSINF, UPV

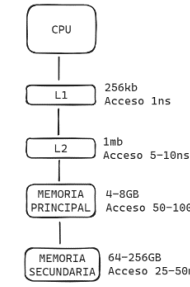
Tema 3.1

Prestaciones del subsistema de Memoria

JERARQUÍA DE MEMORIA

Idealmente se quiere acceder a cantidades de memoria ilimitas super rápido. Pero esto no es posible racionalmente. Pa acercarse a eso se divide la memoria en diferentes niveles, contra más cerca esté del procesador ha de ser + fast.

Principio de localidad: Razón por la que las cachés funcionan. Los programas tienden a reutilizar el código y los datos utilizados recientemente.



- **Localidad temporal**: Los **elementos ya accedidos** serán accedidos nuevamente en el futuro próximo. **Alta en el Código**: "*El 10 % del código se ejecuta durante el 90 % del tiempo de ejecución de un programa*".
- **Localidad espacial**: Los **elementos con direcciones cercanas** tienden a referenciarse cercanamente en el tiempo. → *Organización en bloques. Por ejemplo los vectores.*

Eficiente: **Combina tecnologías con distintas características**. Rápidas pero caras para prestaciones y Densas pero baratas (*coste por bit*) para almacenamiento.

Memoria Cache

Se le llama así a cualquier cos que almacene información que se va a reutilizar. *Sitio seguro donde guardar cosas.*

- **Acierto**: El procesador encuentra el dato en la cache (menor tiempo posible).
- **Fallo**: El procesador NO encuentra el dato en la cache (penalización). Tendrá que esperar a que se **traiga el bloque** de los niveles de memoria inferior.

La gestión de la cache: Aciertos, fallos y reemplazos, se realiza mediante **hardware**.

Memoria Virtual

Si el sistema soporta memoria virtual, los objetos referenciados por un programa deben estar en memoria principal o en disco. El **espacio de direccionamiento** se divide en páginas que deben estar en memoria para ejecutarse.

- **Fallo**: Si se produce un fallo de página, la página entera se transfiere desde el disco a la memoria principal.

Gestión: Ahora la gestión no para el procesador para que espere, se hace por software. *El procesador ejecuta otras cosas mientras espera a que le llegue la cache.*

ESTRUCTURA Y FUNCIONAMIENTO DE LAS CACHES

Cada cache se puede identificar con 4 preguntas:

- **Ubicación de un bloque**: ¿Dónde puede ubicarse un bloque?
- **Identificación de un bloque**: ¿Cómo se encuentra un bloque?
- **Reemplazamiento**: ¿Qué bloque se elimina ante un fallo?
- **Política ante escrituras**: ¿Qué se hace ante una escritura?

Ubicación de un bloque

Correspondencia directa: Un **bloque** **solo puede estar almacenado en una línea** de la cache.

- $\#\boldsymbol{línea} = \#bloque\ mod\ \#\boldsymbol{líneas}\_de\_cache$

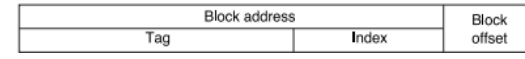
Correspondencia totalmente asociativa: Un **bloque** **puede almacenarse en cualquier lineal** de la cache.

Correspondencia asociativa por conjuntos de n vías: Un **bloque** solo **puede almacenarse en un conjunto** que contiene n líneas de la cache.

- $\#\boldsymbol{conjunto} = \#bloque\ mod\ \#\boldsymbol{conjuntos}\_de\_cache$

Identificación de un bloque

Cada bloque almacenado en la cache tiene asociada una etiqueta que tiene con la dirección real del bloque (pero NO toda igual). Para saber si está en cache, compara la etiqueta de la dirección de bloque con las etiquetas del conjunto donde le toca ir.



Tag: Sirven para comparar y saber si el bloque está. Se hace comparando todos a la vez. A más líneas más caro.

- **Totalmente asociativa**: Pones en el tag los bits de dirección del bloque y comparas con todas las líneas
- **Correspondencia directa**: podrás poner MENOS BITS en el TAG y SOLO tienes que comparar con esa línea
- **Asociativa por conjuntos**: Ahorras algunos bits y solo comparas con las líneas del Conjunto al que vaya.

Bit valid: Un bit "válido" (V) indica si una línea tiene o no información válida. Solo se comparan aquellas etiquetas que tienen el bit valido activo.

Offset: son los bits para seleccionar cada una de las palabras del bloque que hay en esa línea.

Index: Te dice el conjunto al que vas.

Reemplazamiento

Cuando hay un fallo de cache, el bloque referenciado se trae desde los niveles inferiores y el conjunto está lleno.

Con correspondencia directa: **SOLO se puede cambiar** la línea en la que va, no eliges.

Con correspondencia asociativa: **LRU** Menos recientemente usado. **SeudoLRU** Menos costoso que el LRU, prestaciones similares con muchas vías. **Aleatoria** Se elije uno al azar. fácil de implementar. Nice si poca localidad

Política de escritura: *Tras modificar actualizar memoria, cuando y como*

En caso de acierto

- **Write-through**: Se **escribe** tanto en la **cache** como en **MP**. Mas fácil de implementar. La memoria principal siempre esta actualizada.
- **Write-back**: **Solo** se escribe en la **cache**. Los bloques "sucios" (*bit dirty activo*) se **actualizan en MP** **cuando se reemplazan**. *Emplea menos ancho de banda de memoria que Write-through.*

Caso de fallo

- **Write allocate**: El **bloque se trae a la cache**. **después**, Según sea Through o back haces una cosa o la otra. *Habitual con write-back*.
- **No-write allocate**: El **bloque no se trae a la cache**. Solo se modifica el nivel inferior. *Usual en write-through*.

PRESTACIONES DEL SUBSISTEMA DE MEMORIA

Tiempo de acceso medio: $T_{acceso} = TA + TF \times PF$

Ta: Tiempo de acierto en cache **TF**: Tasa de fallos **PF**: Penalización de fallo.

Modificación de la ecuación del tiempo de ejecución: Ahora le añades el tiempo extra de acceder a memoria.

$$Tej = T_{ejCPU} + T_{extraMemoria} = I \cdot CPI \cdot T + I \cdot API \cdot TF \cdot PF \cdot T$$

$$T_{extraMemoria} = Ciclos\ extra \cdot T = NFallos \cdot PFallos \cdot T = I \cdot \frac{Accesos}{I} \cdot \frac{NF}{Accesos} \cdot PF \cdot T = I \cdot API \cdot TF \cdot PF \cdot T$$

API: 1 para las instrucciones (cuando hacen IF) + % de las loads y strores. *Si el 25% de instruc son load y estores = 1+0,25 = 1,25*

COSA: Directa más fallos que Asociativa que más fallos totales. Tiempo ciclo menor en directa que asociativa que total (retardo de comparar más y del mux)

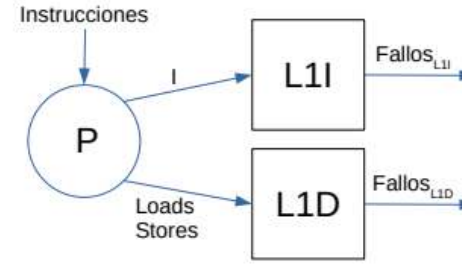
Cache separadas de instrucciones y datos

Ahora los ciclos de parada serán: Ciclos de parada debido a instrucciones + Ciclos de parada debido a Datos =

$$NF^i \cdot PF \cdot T + NF^d \cdot PF \cdot T =$$

$$I \cdot \frac{Accesos^I}{I} \cdot \frac{NF^i}{Accesos^i} \cdot PF^i \cdot T = I \cdot API^i \cdot TF^i \cdot PF \cdot T$$

$$I \cdot \frac{Accesos^d}{I} \cdot \frac{NF^d}{Accesos^d} \cdot PF^d \cdot T = I \cdot API^d \cdot TF^d \cdot PF \cdot T$$



Tiempo de parada: $I \cdot API^i \cdot TF^i \cdot PF \cdot T + I \cdot API^d \cdot TF^d \cdot PF \cdot T = I \cdot \left(API^i \cdot TF^i + API^d \cdot TF^d\right) \cdot PF \cdot T = I \cdot API \cdot \frac{\left(API^i \cdot TF^i + API^d \cdot TF^d\right)}{API} \cdot PF \cdot T$

- $API = API^d + API^i$
- TF UNIFICADA (si no nos dicen que está separada): $\frac{\left(API^i \cdot TF^i + API^d \cdot TF^d\right)}{API}$

Tiempo de ejecución: $CPI \cdot I \cdot T + I \cdot API^i \cdot TF^i \cdot PF \cdot T + I \cdot API^d \cdot TF^d \cdot PF \cdot T$

Diferente penalización de fallo por instrucción

Cuando cada instrucción tarda tiempo diferente al acceder a la cache:

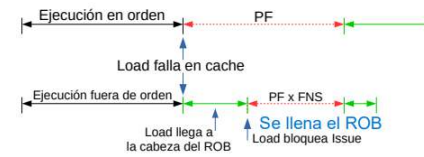
- Las API de cada uno son los: $\frac{I}{I}, \frac{Loads}{I}, \frac{Stores}{I}$
- API global: $API^I(1) + API^{Load} + API^{Store}$

$$T_{\text{extra mem.}} = I \times \frac{I}{I} \times TF_{LII} \times PF_{LII} \times T +$$
$$I \times \frac{\text{Loads}}{I} \times TFL_{LID} \times PFL_{LID} \times T +$$
$$I \times \frac{\text{Stores}}{I} \times TFE_{LID} \times PFE_{LID} \times T$$

Procesador con ejecución fuera de orden: *o-o-o*

Búsqueda de instrucción: Los fallos en la cache de instrucciones detienen la búsqueda de instrucciones. *SI o SÍ CILOS DE PARAD, TE ESPERAS, por lo que TF = 100% .*

Lecturas: Ante un fallo de lectura, el front-end (los IFs) no se detiene hasta que no haya una entrada disponible en ROB o estaciones de reserva/buffers. → $PFL_{L1D}\ no\ solapada\ =\ PFL_{L1D}\ \times\ FNS$ (Fraccion No Solapada). *Te los comes parcialmente según tengas sitio en el ROB.*



Escrituras: Las escrituras se realizan en buffers de escritura y sus fallos practicamente no detienen el front-end: $PFE_{L1D}\ \approx\ 0$. *Sigues haciendo cosas has que se llenen los buffers de escritura, pero supones que siempre hay*. NO SE TIENE EN CUENTA.

Conclusión:

$$T_{\text{extra mem.}} = \mathrm{I} \times \mathrm{TF_{LII}} \times \mathrm{PF_{LII}} \times \mathrm{T} + \mathrm{I} \times \frac{\text{Loads}}{\mathrm{I}} \times \mathrm{TFL_{LID}} \times \mathrm{PFL_{LID}} \times \mathrm{FNS} \times \mathrm{T}$$