ARCHITETTURA DEGLI ELABORATORI

A.A. 2020-2021

Università di Napoli Federico II
Corso di Laurea in Informatica

Docenti
Proff. Luigi Sauro gruppo 1 (A-G)
Silvia Rossi gruppo 2 (H-Z)

MEMORIE

Il bit di validità

- Un ulteriore bit di controllo, chiamato ***bit di validità***, è necessario per ogni blocco.
- **Questo bit indica se il blocco contiene o meno dati validi**.
- Non deve però essere confuso con il ***bit di modifica***, menzionato in precedenza, il bit di modifica, che indica se il blocco è stato modificato o meno durante il periodo in cui è stato nella cache, serve soltanto nei sistemi che non fanno uso del metodo ***write-through***.

I bit di validità dei blocchi che si trovano nella cache vengono tutti posti a 0 nell'istante iniziale di accensione del sistema e, in seguito, ogni volta che nella memoria principale vengono caricati programmi e dati nuovi dal disco.

Il Direct Memory Access - DMA

I trasferimenti dal disco alla memoria principale vengono effettuati mediante un meccanismo detto di DMA (Direct Memory Access). Questa sigla specifica che durante il trasferimento la CPU non interviene, perché è la logica di controllo del disco che gestisce le linee indirizzo e le linee dati, fornendo i segnali di controllo necessari.

Come vedremo successivamente, in una operazione di questo genere vengono trasferite grandi quantità di istruzioni e dati organizzati in entità dette "***pagine***" ciascuna delle quali di solito contiene migliaia di ***locazioni*** e quindi centinaia di ***blocchi***.
Normalmente, le pagine di programma e quelle di dati vanno nella memoria centrale senza coinvolgere la cache.

Gestione del bit di validità

Il bit di validità di un certo blocco della cache viene posto a "1" la prima volta che esso è chiamato a contenere istruzioni o dati trasferiti dalla memoria principale; poi, ogni volta che le locazioni di un blocco della memoria principale vengono interessate da un trasferimento di nuove istruzioni o di nuovi dati dal disco, viene effettuato un controllo per determinare se qualcuno dei blocchi che stanno per essere sovrascritti fossero o meno presenti nella cache.

Se nella cahe c'è una copia del blocco, il suo bit di validità viene posto a "0"; in tal modo, si garantisce che nella cache non ci siano dati obsoleti.

Il bit di validità a "0" candida immediatamente il blocco ad essere sovrascritto.

Svuotamento (flush) della cache

Una situazione dello stesso tipo si determina quando viene effettuato un trasferimento tramite DMA dalla memoria al disco e la cache utilizza un protocollo write-back.
In questo caso, i dati nella memoria potrebbero non riflettere i cambiamenti che possono essere stati fatti sulla copia dei dati presenti nella cache.

Una possibile soluzione a tale problema consiste nello ***svuotamento*** (***flush***) della cache forzando i dati con il bit di modifica attivo a essere ricopiati nella memoria principale prima che venga effettuato il trasferimento tramite DMA.

“Coerenza” della cache

Il sistema operativo è in grado di fare tutto tutte le operazioni necessarie in modo molto efficiente, senza peggiorare significativamente le prestazioni, visto che le operazioni di ingresso/uscita di scrittura non sono molto frequenti.

Questa necessità di garantire che due diverse entità (i sottosistemi della CPU e del DMA nel caso presente) utilizzino la stessa copia dei dati viene chiamata problema della ***coerenza della cache***.

Algoritmi di sostituzione (1 / 2)

- In una cache ad indirizzamento diretto, la posizione di ogni blocco è predefinita, quindi non esiste alcuna strategia di sostituzione.

- Nelle memorie cache ***associative*** e ***set-associative***, esiste invece una certa flessibilità.

Quando un nuovo blocco deve essere portato nella cache, e tutte le posizioni che potrebbe occupare contengono dati validi, il controllore della cache deve decidere quale blocco, tra quelli esistenti, soprascrivere.

Questa è una decisione importante, perché la scelta potrebbe essere determinante per le prestazioni del sistema.

Algoritmi di sostituzione (2 / 2)

In generale, l'obiettivo è quello di **mantenere nella cache quei blocchi che hanno una maggior possibilità di essere nuovamente utilizzati nel prossimo futuro**. ma, non è facile prevedere i blocchi a cui si farà riferimento.

Una possibile strategia è di tener presente che la ***località dei riferimenti*** suggerisce che i blocchi a cui c'è stato accesso di recente hanno elavata probabilità di essere utilizzati nuovamente entro breve tempo.

Sostituzione con algoritmo LRU

Quando bisogna eliminare dalla cache un blocco, **è sensato soprascrivere quello a cui non si accede da più tempo**. Tale blocco prende il nome di blocco ***utilizzato meno di recente*** (***Least Recently Used***, **LRU**), e la tecnica si chiama ***algoritmo di sostituzione LRU***.

Per utilizzare l'algoritmo LRU, il controllore della cache deve mantenere traccia di tutti gli accessi ai blocchi mentre l'elaborazione prosegue.

Il blocco LRU di una memoria cache ***set-associativa*** con insiemi di quattro blocchi, può, ad esempio, essere indicato da un contatore di 2 bit associato a ciascun blocco, il cui contenuto sia opportu-namente gestito secondo un protocollo automatico

Gestione del contatore di accessi

Quando si ha un successo nell'accesso al blocco (*hit*):

- il contatore di quel blocco viene posto a 0. I contatori con valori originariamente inferiori a quelli del blocco a cui si accede, vengono incrementati di uno, mentre tutti gli altri rimangono invariati.

Quando, invece, l'accesso fallisce (*miss*) si aprono due possibilità:

- l'insieme non è pieno, il contatore associato al nuovo blocco caricato dalla memoria principale viene posto a 0 e il valore di tutti gli altri contatori viene incrementato di l.
- l'insieme è pieno, si rimuove il blocco, il cui contatore ha valore 3, e si pone il nuovo blocco al suo posto, mettendo il contatore a 0. Gli altri tre contatori dell'insieme vengono incrementati di un'unità.

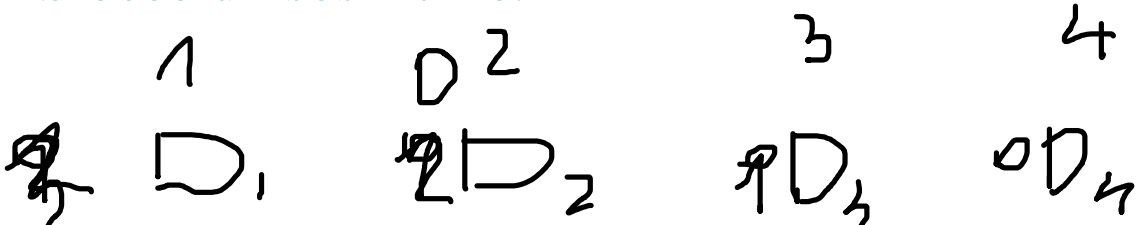
E' facile verificare che i valori dei contatori dei blocchi occupati sono sempre diversi.

Alternative all'algoritmo LRU

L'algoritmo LRU è stato utilizzato ampiamente, con buone prestazioni, in numerosi schemi di accesso. Ma in alcune situazioni, per esempio nel caso di accessi sequenziali a un vettore di elementi che è leggermente troppo grande per poter stare tutto nella cache, le prestazioni sono molto scadenti

Le prestazioni dell'algoritmo LRU possono essere migliorate introducendo una piccola dose di casualità nella scelta del blocco da sostituire.

Sono stati proposti molti altri algoritmi di sostituzione più o meno complessi, che richiedono talvolta anche notevoli complicazioni hardware, ma l'algoritmo più semplice, e spesso anche piuttosto efficace, consiste nello **scegliere in modo completamente casuale il blocco da sostituire**.



Types of Misses

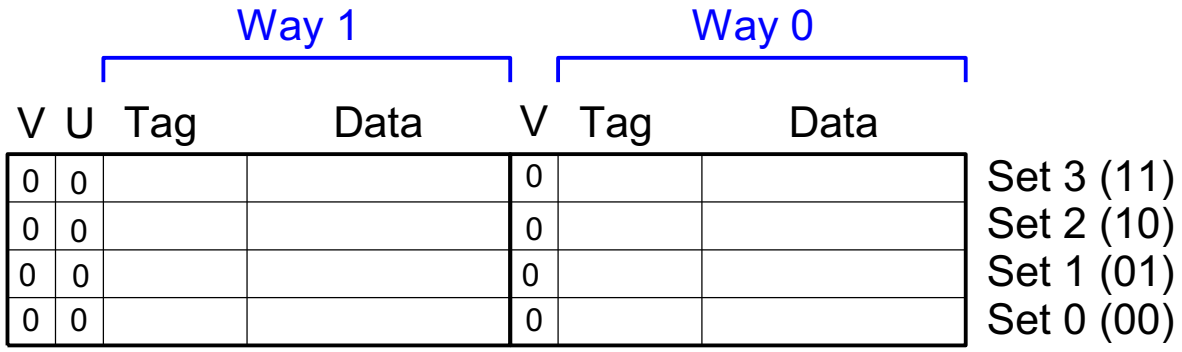
- **Compulsory:** first time data accessed
- **Capacity:** cache too small to hold all data of interest
- **Conflict:** data of interest maps to same location in cache

Miss penalty: time it takes to retrieve a block from lower level of hierarchy

LRU Replacement

ARM Assembly Code

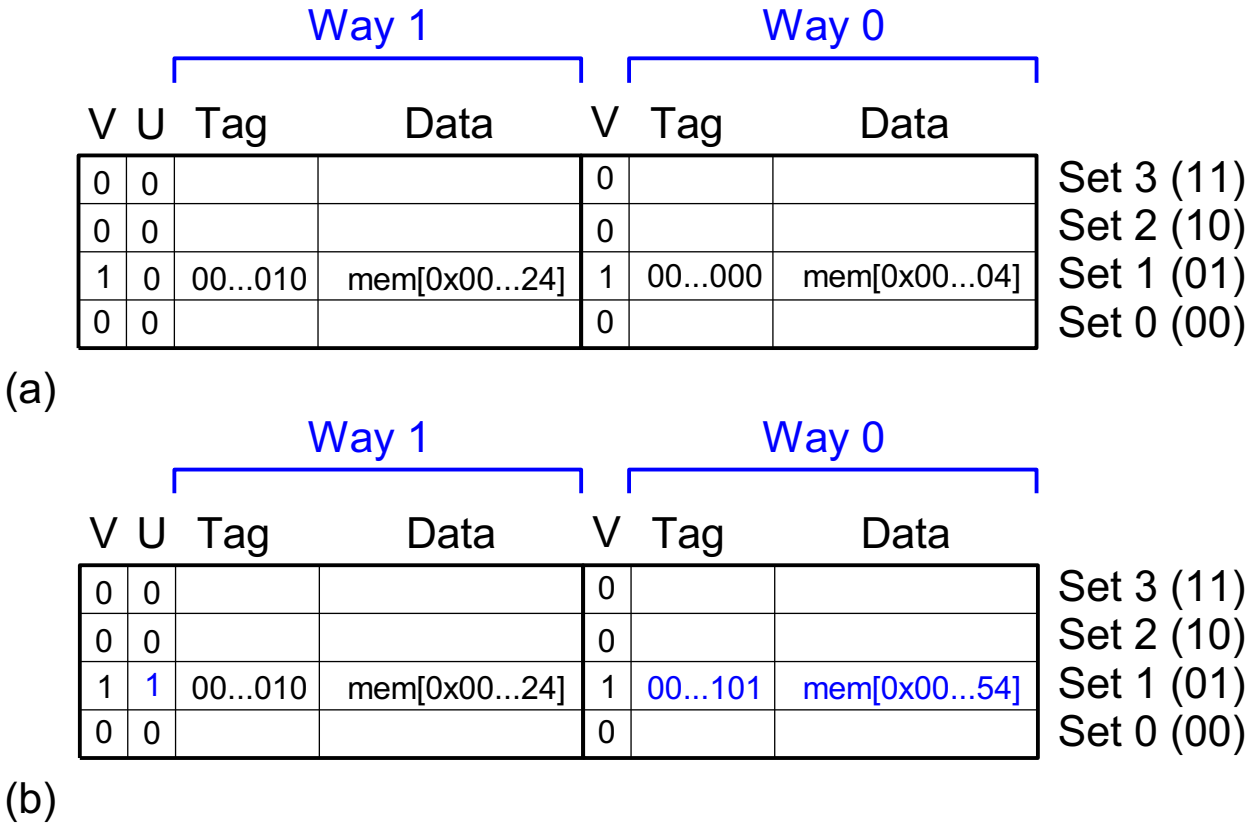
```
MOV R0, #0
LDR R1, [R0, #4]
LDR R2, [R0, #0x24]
LDR R3, [R0, #0x54]
```



LRU Replacement

ARM Assembly Code

```
MOV R0, #0
LDR R1, [R0, #4]
LDR R2, [R0, #0x24]
LDR R3, [R0, #0x54]
```

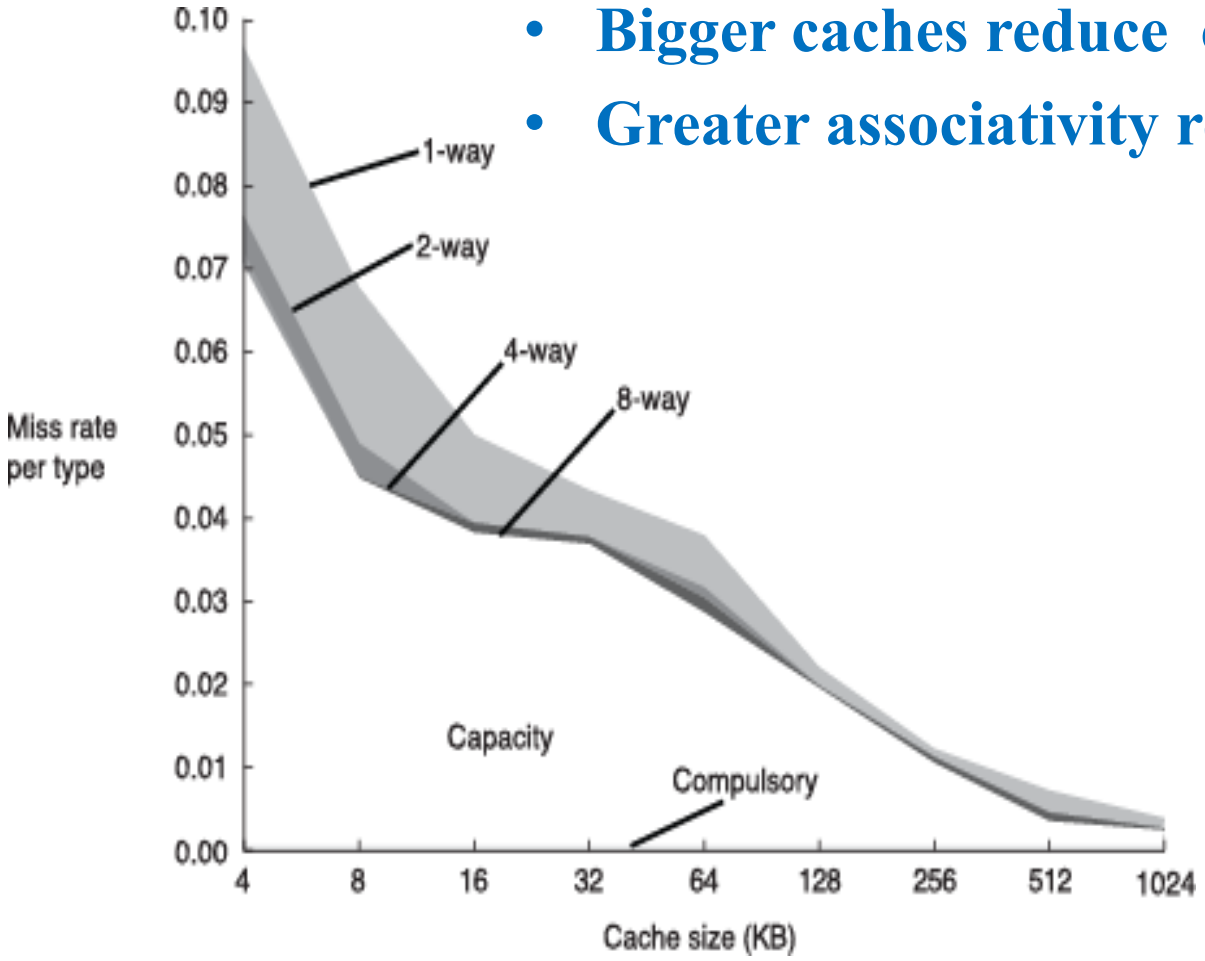


Cache Summary

- **What data is held in the cache?**
  - Recently used data (temporal locality)
  - Nearby data (spatial locality)
- **How is data found?**
  - Set is determined by address of data
  - Word within block also determined by address
  - In associative caches, data could be in one of several ways
- **What data is replaced?**
  - Least-recently used way in the set

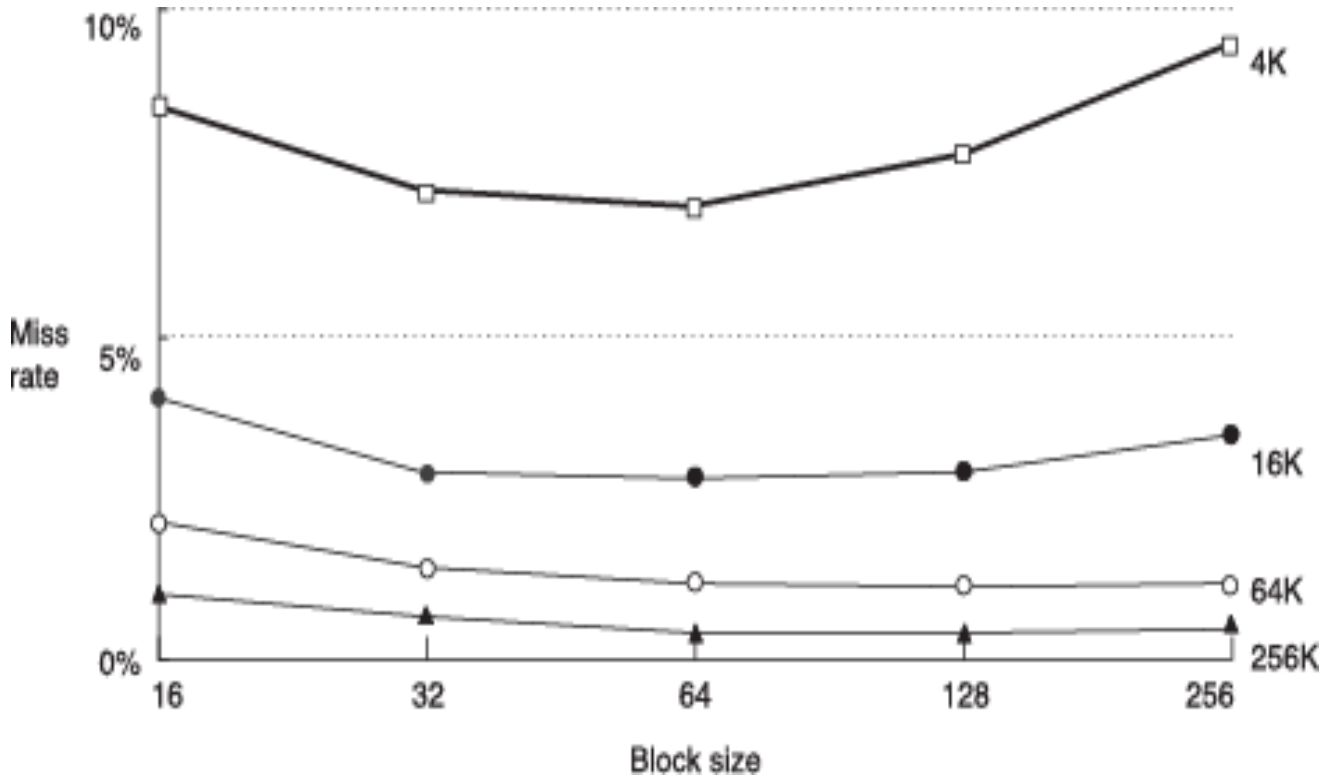
Miss Rate Trends

- **Bigger caches reduce capacity misses**
- **Greater associativity reduces conflict misses**



Adapted from Patterson & Hennessy, *Computer Architecture: A Quantitative Approach*, 2011

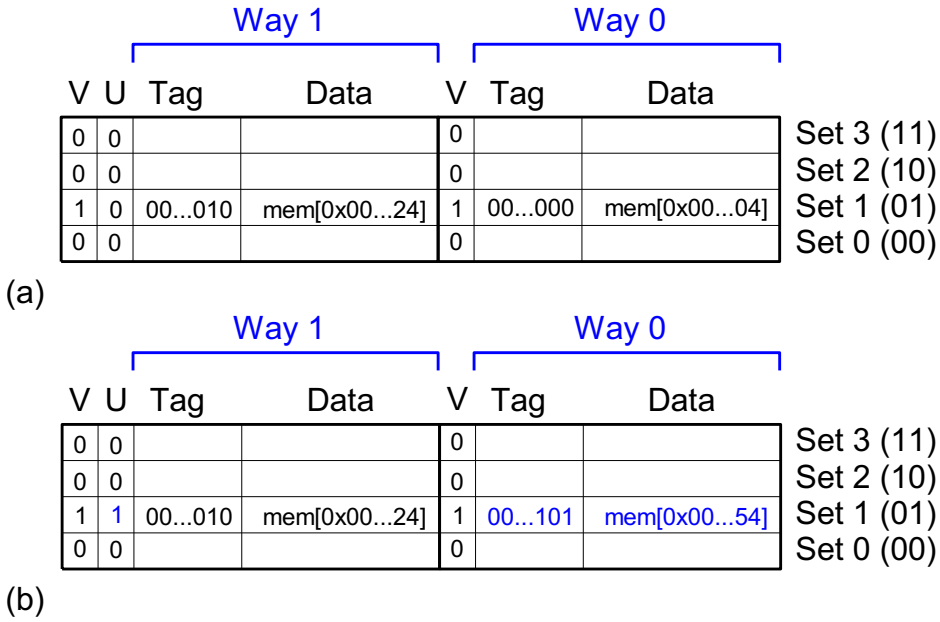
Miss Rate Trends



- **Bigger blocks reduce compulsory misses**
- **Bigger blocks increase conflict misses**
  - Data una cache di dimensioni fissate, all'aumentare del block size il numero di set decrementa (SxNxb=C). Quindi aumentano i miss di conflitto

ARM Assembly Code

```
MOV R0, #0
LDR R1, [R0, #4]
LDR R2, [R0, #0x24]
LDR R3, [R0, #0x54]
```



Multilevel Caches

- Larger caches have lower miss rates, longer access times
- Expand memory hierarchy to multiple levels of caches
  - Level 1: small and fast (e.g. 16 KB, 1 cycle)
  - Level 2: larger and slower (e.g. 256 KB, 2-6 cycles)
- Most modern PCs have L1, L2, and L3 cache

Una cache per le istruzioni ed una per i dati?

Per i motivi che abbiamo più volte ricordato **il posto migliore in cui collocare una cache è il chip della CPU**. Sfortunatamente, però, si trova posto solo per cache piccole.
Tutti i processori con prestazioni elevate hanno una cache “on chip”.

Alcuni produttori hanno deciso di realizzare due cache separate, una per le istruzioni e un'altra per i dati, come nei processori 68.040 e PowerPC 604. In altri casi è stata adottata una sola cache per istruzioni e dati, come nel processore PowerPC 601.

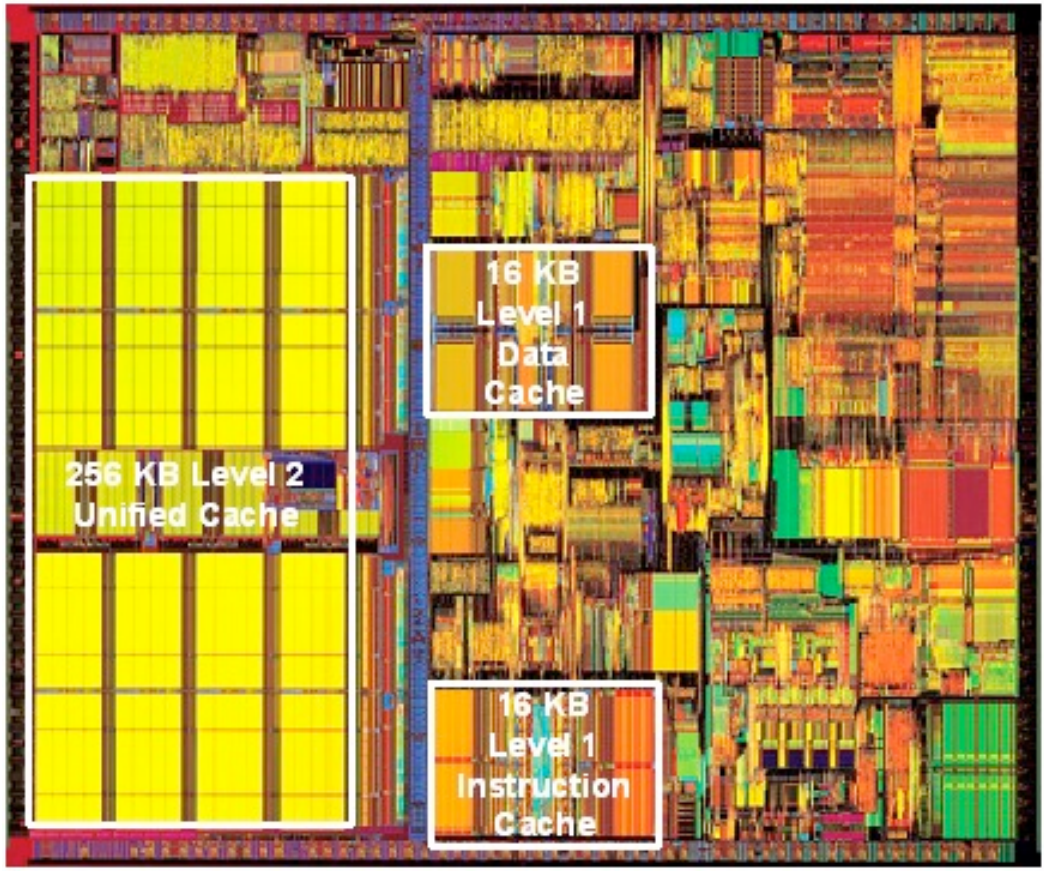
Una cache per le istruzioni ed una per i dati?

Una sola cache per istruzioni e dati in teoria dovrebbe avere una frequenza di successo superiore, poiché offre una maggiore flessibilità nella memorizzazione di nuove informazioni.

Tuttavia, se si utilizzano cache separate, è possibile accedere contemporaneamente a entrambe le memorie cache, aumentando così il parallelismo e, di conseguenza, le prestazioni della CPU.

Lo svantaggio delle cache separate è che l'incremento del grado di parallelismo è accompagnato da circuiti molto più complessi.

Intel Pentium III Die



Cache multi livello

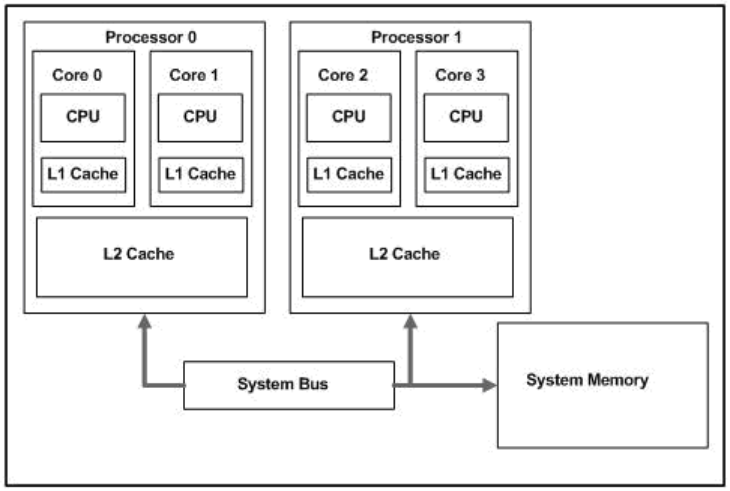
- Oggigiorno la memoria cache costituisce una gerarchia di 2 o 3 livelli: L1, L2, L3
- Vi possono essere due tipologie di cache multi livello: inclusiva o esclusiva
- Inclusiva: L2 contiene i dati presenti in L1 (similmente L2 è contenuta in L3)
  - L2 contiene sia i dati correnti di L1, sia quelli che erano presenti in precedenza e sono stati sovrascritti mediante l'algoritmo di sostituzione (copy back)
  - Per mantenere la coerenza L1 ha una politica di write-through verso L2, e così L2 verso L3
  - Miss in L1 e hit in L2: il blocco relativo viene copiato da L2 a L1 (in caso di sostituzione un dato va da L2 a L1 e un'altro da L1 a L2)
  - Miss in L1 e L2: il blocco dalla main memory viene copiato sia in L1 che L2

Cache multi livello

- Oggigiorno la memoria cache costituisce una gerarchia di 2 o 3 livelli: L1, L2, L3
- Vi possono essere due tipologie di cache multi livello: inclusiva o esclusiva
- Esclusiva: L2 contiene solo i dati di copy back da L1 (victim cache)
  - La memoria complessiva della cache è data dalla somma delle capacità dei vari livelli, in realtà nelle moderne architetture la capacità di L2 è un ordine di grandezza rispetto a quella di L1, e così fra L3 e L2, il guadagno rispetto alle cache inclusive non è molto significativo).
  - Miss in L1 e hit in L2: Le linee di cache di L1 e L2 vengono scambiati fra loro, cioè la linea di cache di L1 viene memorizzata in L2 e la linea di L2 in L1.
  - Miss in L1 e L2: Il dato letto dalla memoria è memorizzato direttamente in L1 e la linea di cache rimpiazzata di L1 (*victim data*) è trasferita in L2 rimpiazzando un'altra linea di cache secondo la politica di rimpiazzo usata.

Cache condivise

- Nelle architetture multi core L2 o L3 possono essere condivisi fra più core di uno stesso processore, questo consente:
- Un uso più efficiente del livello condiviso
  - Se un core è inattivo, allora l'altro può utilizzare per se il livello condiviso
- Nella programmazione multi-threading consente di utilizzare in maniera più efficiente dati condivisi
  - Un core può eseguire un pre-/post-processing di dati forniti dall'altro core
- Riduce il front-side bus traffic
  - Un unico livello condiviso che si interfaccia con la main memory



Esercizi sulle cache

Domanda: Si ipotizzi che il 30% delle istruzioni in un programma tipico effettui un'operazione di scrittura o di lettura in RAM, e che la frequenza di successo di lettura in cache sia del 95% per le istruzioni e del 90% per i dati. Si supponga, inoltre, che la penalità di fallimento sia la stessa per operazioni di scrittura e operazioni di lettura. Si supponga che, in cicli di clock, una lettura diretta in memoria costi 10, una in cache costi 1 e una sia in cache che in memoria costi 16. Quale sarebbe il guadagno utilizzando la cache?

Esercizi sulle cache

Domanda: Si ipotizzi che il 30% delle istruzioni in un programma tipico effettui un'operazione di scrittura o di lettura in RAM, e che la frequenza di successo di lettura in cache sia del 95% per le istruzioni e del 90% per i dati. Si supponga, inoltre, che la penalità di fallimento sia la stessa per operazioni di scrittura e operazioni di lettura. Si supponga che, in cicli di clock, una lettura diretta in memoria costi 10, una in cache costi 1 e una sia in cache che in memoria costi 16. Quale sarebbe il guadagno utilizzando la cache?

Date n istruzioni, costo senza cache è: (1 + 0,3) n 10
Costo con cache: n (0,95 * 1 + 0,05 * 16) + 0,3 n (0,9 * 1+0,1 * 16)

$$G=\frac{(1{,}3*10)n}{[(0{,}95+16*0{,}05)+(0{,}9+0{,}1*16)]n}=\frac{13}{2.5}=5.2$$

6,95

1,87

* 0,3