La gerarchia di memorie (2)

Architetture Avanzate dei Calcolatori

Valeria Cardellini

Migliorare le prestazioni delle cache

- Consideriamo la formula del tempo medio di accesso in memoria (AMAT)

  AMAT = hit time + miss rate $\times$ miss penalty

- Come possiamo migliorare l'AMAT?
  - Riducendo i fattori presenti nella formula: hit time, miss rate e miss penalty
- Analizziamo quattro categorie di ottimizzazioni delle cache
  - Riduzione del miss penalty
  - Riduzione del miss rate
  - Riduzione del miss penalty *e* del miss rate tramite parallelismo
  - Riduzione dell'hit time

Ridurre il miss penalty

- Esaminiamo le seguenti soluzioni per ridurre il miss penalty
  - Cache multi-livello
  - Critical word first, early restart
  - Priorità ai read miss rispetto alle scritture
  - Fusione sul write buffer
  - Victim cache

Cache multi-livello

- Gerarchia composta da due o più livelli di cache
  - Cache più piccole e veloci vicino al processore
  - Cache più grandi e lente scendendo nella gerarchia
  - Anche 3 livelli di cache nei processori moderni
- Nel caso di 2 livelli (L1 e L2):
  - Cache di primo livello abbastanza piccola da essere molto veloce
  - Cache di secondo livello abbastanza grande da soddisfare molti degli accessi che altrimenti andrebbero in DRAM

$$\text{AMAT} = \text{hit time}_{L1} + \text{miss rate}_{L1} \times \text{miss penalty}_{L1}$$

$$\text{miss penalty}_{L1} = \text{hit time}_{L2} + \text{miss rate}_{L2} \times \text{miss penalty}_{L2}$$

$$\text{AMAT} = \text{hit time}_{L1} + \text{miss rate}_{L1} \times (\text{hit time}_{L2} + \text{miss rate}_{L2} \times \text{miss penalty}_{L2})$$

  - miss rate_{L2} si misura sugli accessi alla cache L1 che non hanno avuto successo

Cache multi-livello (2)

- ***Local miss rate*** (frequenza di fallimento locale): numero di miss in una cache diviso il numero totale di accessi *alla stessa cache*
  - Per cache L1: local miss rate$_{L1}$ = miss rate$_{L1}$
  - Per cache L2: local miss rate$_{L2}$ = miss rate$_{L2}$
- ***Global miss rate*** (frequenza di fallimento globale): numero di miss nella cache diviso il numero totale di accessi *generato dal processore*
  - Per cache L1: global miss rate$_{L1}$ = miss rate$_{L1}$
  - Per cache L2: global miss rate$_{L2}$ = miss rate$_{L1}$ × miss rate$_{L2}$

Ridurre il miss penalty con cache multi-livello

- La cache di secondo livello ha senso se è molto più grande della cache di primo livello
- La velocità della cache di primo livello influenza il ciclo di clock del processore
- La velocità della cache di secondo livello influenza solo il miss penalty della cache di primo livello (miss penalty$_{L1}$)
- Per la cache di secondo livello si può usare un'organizzazione semplice (es. indirizzamento diretto)
- I dati presenti nella cache di primo livello sono presenti anche nella cache di secondo livello? Due soluzioni:
  - *Multi-level inclusion*: i dati in L1 sono *sempre* presenti in L2
    - Si mantiene la consistenza tra cache e I/O
  - *Multi-level exclusion*: i dati in L1 non sono *mai* presenti in L2
    - Soluzione ragionevole per cache L2 di piccole dimensioni

Early restart e critical word first

- **Early restart**
  - Miss penalty = latenza per la prima parola + tempo di trasferimento per il resto del blocco
  - Con l'early restart si riprende l'esecuzione non appena arriva la parola richiesta (senza aspettare tutto il blocco)
  - Efficace per accessi ad istruzioni
- **Critical word first** (anche detta *requested word first*)
  - In presenza di read miss, la parola mancante viene richiesta alla memoria di livello inferiore e inviata al processore non appena arriva
  - Il processore inizia l'esecuzione mentre si riempie anche il resto del blocco
- Queste tecniche sono utili per ridurre il miss penalty solo se i blocchi sono grandi

Priorità ai read miss rispetto alle scritture

- Cache di tipo write-through con write buffer
  - Il write buffer potrebbe contenere il valore di una parola richiesta in caso di read miss
  - Soluzione più semplice: un read miss attende che il write buffer sia vuoto (lenta!)
  - Soluzione più veloce: in caso di read miss, si controlla il contenuto del write buffer; se non c'è conflitto, si serve il read miss (che riceve priorità sulla scrittura)
- Cache di tipo write-back
  - Il read miss può rimpiazzare un blocco dirty
  - Si copia il blocco dirty in un buffer, si effettua la lettura dalla memoria di livello inferiore, infine si scrive il blocco dirty dal buffer in memoria

Fusione sul write buffer

- Si usa il write buffer anche per cache write-back
- Se il write buffer è vuoto:
  - Vi si scrivono i dati e l'indirizzo completo; dal suo punto di vista, il processore ha finito la scrittura
- Se il write buffer contiene blocchi modificati:
  - Si controlla l'indirizzo dei dati da memorizzare; se il nuovo indirizzo è uguale a quello di un elemento valido nel write buffer, i nuovi dati vengono combinati con tale elemento (*write merging*)
- Se il write buffer è pieno e non ci sono coincidenze di indirizzo:
  - Il processore aspetta finché nel write buffer non si libera una posizione

Victim cache

- Idea base: ciò che viene scartato potrebbe essere di nuovo utile a breve
- Si inserisce una piccola cache associativa (detta *victim cache*) fra la cache ed il suo percorso di riempimento
  - La victim cache contiene solo i blocchi scartati dalla cache in caso di miss
  - In caso di miss, i blocchi scartati e messi nella victim cache vengono verificati prima di passare alla memoria di livello inferiore
  - Se il blocco richiesto è nella victim cache, si scambia questo blocco con quello in cache
- La victim cache è particolarmente utile per ridurre il miss penalty causato dai miss di conflitto

Ridurre il miss rate

- Per ridurre il miss rate si può:

Soluzioni già esaminate

  - Aumentare la dimensione del blocco
    - Si sfrutta il principio di località spaziale e si riducono i compulsory miss (ma aumenta il miss penalty)
  - Aumentare la dimensione della cache
    - Si riducono i capacity miss ed i conflict miss (ma aumenta l'hit time ed il costo)
  - Aumentare il grado di associatività
    - Si riducono i conflict miss (ma aumenta l'hit time)
  - Ottimizzazioni del compilatore

Ottimizzazioni del compilatore

- Le altre tecniche per ridurre il miss rate richiedono modifiche o aggiunte all'hardware
- Ottimizzazioni del codice effettuate durante la compilazione permettono di ridurre il numero di miss senza modificare l'hardware
  - Ottimizzazioni orientate a risolvere separatamente instruction miss e data miss
  - Le tecniche di ottimizzazione attuate dal compilatore riorganizzano il codice in modo da migliorare la località temporale e/o spaziale
- Analizziamo due tecniche per ridurre il *data miss rate*:
  - **Loop interchange**
  - **Blocking**

Loop interchange

- Obiettivo del **loop interchange**: migliorare la *località spaziale*
- Si scambia l'ordine di cicli annidati in modo che si acceda ai dati nell'ordine in cui questi sono memorizzati
- Esempio
  - In C le matrici (array bidimensionali) sono memorizzate secondo il *row major order*

Loop interchange

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

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

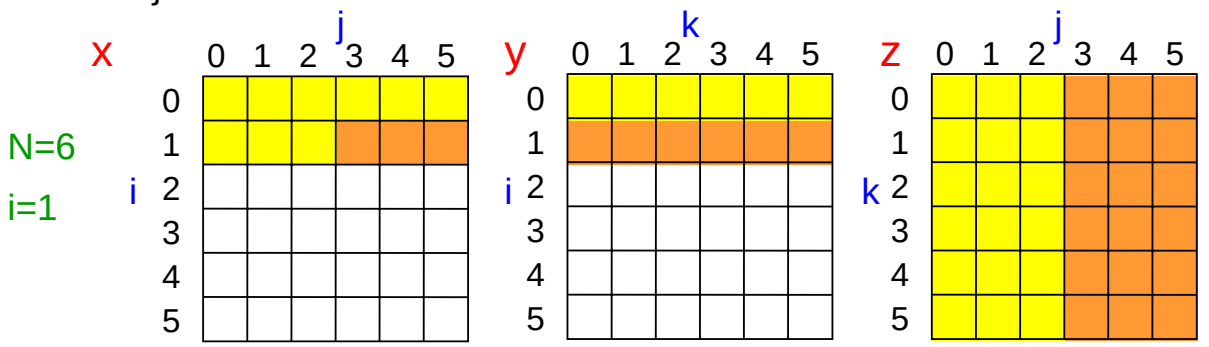

Blocking

- Obiettivo del **blocking**: migliorare la *località temporale*
- Per un algoritmo in cui ad alcune matrici si accede per righe e ad altre per colonne, memorizzare le matrici per righe piuttosto che per colonne non migliora le prestazioni
- Con il blocking, invece di operare su righe o colonne intere, l'algoritmo opera su sottomatrici (*blocchi*): lo scopo è massimizzare l'accesso ai dati in cache prima che vengano sostituiti

Blocking (2)

- Esempio: moltiplicazioni tra due matrici quadrate y e z

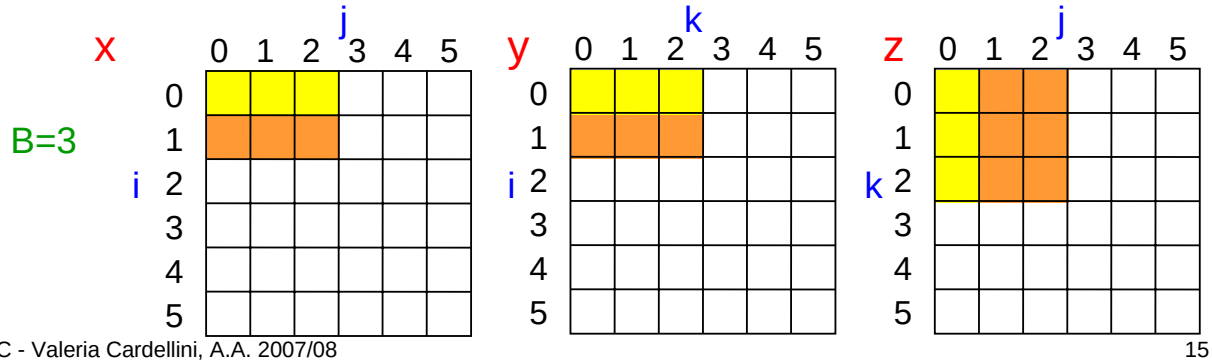
```
/* Before */
for (i=0; i<N; i++)
        for (j=0; j<N; j++) {
                r = 0;
                 for (k=0; k<N; k++)
                        r += y[i][k]*z[k][j];
                x[i][j] = r;
        }
```



Blocking (3)

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

- Il codice lavora su una sottomatrice di dimensione BxB (B è detto *fattore di blocking*)



Ridurre il miss penalty e miss rate tramite parallelismo

- Principio: sovrapporre l'esecuzione del programma all'attività della gerarchia di memorie
- A volte gli approcci sono ottimizzati per le specifiche architetture di processori
- Soluzioni esaminate
  - Cache non bloccanti
  - Prefetching hardware di istruzioni e dati
  - Prefetching controllato dal compilatore

Cache non bloccanti

- Il processore continua a leggere istruzioni dalla I-cache durante un D-cache miss
- Più in generale: la D-cache continua a fornire hit anche durante un miss ("hit under miss")
- E' una tecnica che ha senso per processori dotati di parallelismo intrinseco
- La complessità del controllore della cache aumenta

Prefetching hardware di istruzioni e dati

- Idea base: lettura *anticipata* (*prefetch*) di elementi (istruzioni o dati) prima che siano richiesti dal processore
- Prefetching di istruzioni
  - Spesso fatto in hardware
  - In caso di miss, il processore legge il blocco richiesto e quello immediatamente consecutivo. Il blocco richiesto viene portato in cache, quello anticipato viene posto nell'*instruction stream buffer* (ISB)
  - Se un blocco richiesto è presente nell'ISB, lo si legge dall'ISB, si cancella la richiesta alla cache, si invia la prossima richiesta di prefetch
- Prefetching di dati
  - In modo simile a quello delle istruzioni

Prefetching controllato dal compilatore

- Il compilatore inserisce istruzioni di prefetch per richiedere i dati prima che occorrano
  - Register prefetch: carica il valore in un registro
  - Cache prefetch: carica i dati in cache

Ridurre l'hit time

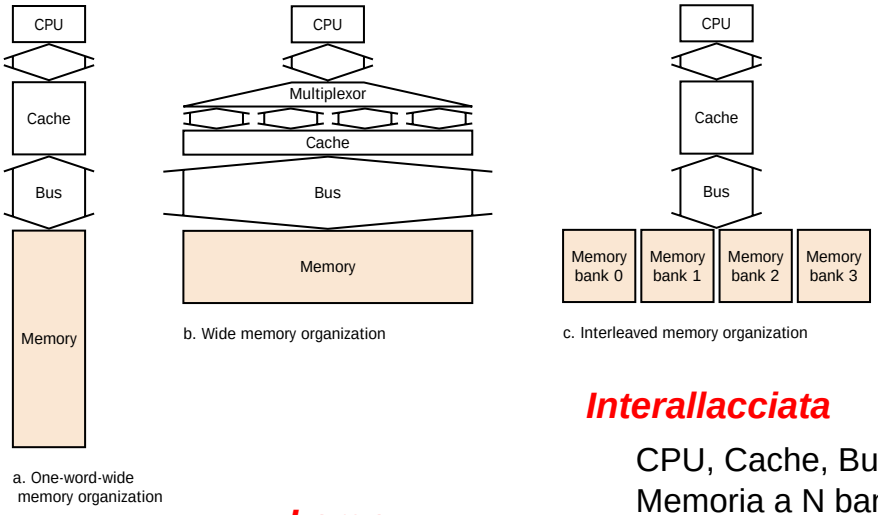
- Per ridurre l'hit time si può:
  - Usare cache piccole e semplici
    - Cache L2 di dimensioni contenute è integrabile sul chip
    - Un'organizzazione semplice (es. mappatura diretta) implica un tempo di accesso inferiore (occorre compiere un minor numero di operazioni)
  - Accesso alla cache in pipeline
    - Aumenta il numero di stadi della pipeline
    - Esempio: nel Pentium 4, occorrono 4 cicli di clock per accedere alla I-cache
  - Trace cache
    - Soluzione adottata nel Pentium 4 per evitare di non dover decodificare in micro-istruzioni una istruzione già incontrata durante l'esecuzione del programma
    - Soluzione non più usata nelle architetture Intel Core

Progettare il sistema di memoria per supportare le cache

- Si può ridurre il miss penalty aumentando la larghezza di banda del bus tra DRAM e cache, in modo tale da poter usare blocchi di grandi dimensioni
- Tre organizzazioni possibili del sistema di memoria
  - Semplice
  - Larga
  - Interallacciata (interleaved)
- Obiettivo: aumentare la larghezza di banda (fisicamente o logicamente)

Come aumentare la larghezza di banda della memoria

- Tre organizzazioni del sistema di memoria



Semplice

CPU, Cache, Bus, Memoria: stessa larghezza (es. 32 bit)

Larga

CPU/Mux: 1 parola
Mux/Cache, Bus, Memoria: N parole
(es. Alpha: 64 bit e 256 bit)

Interallacciata

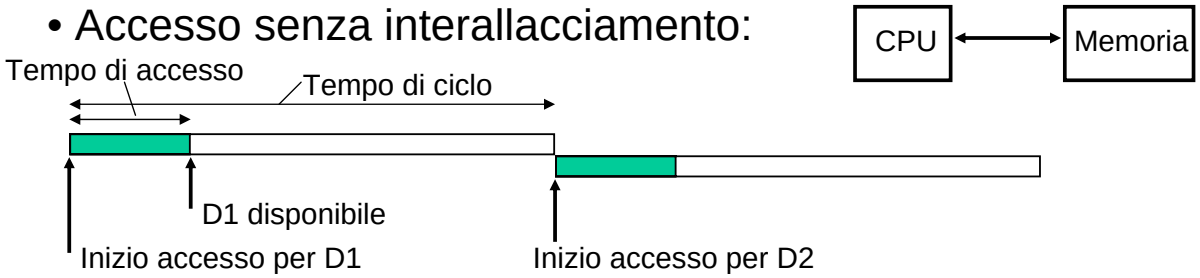
CPU, Cache, Bus: 1 parola
Memoria a N banchi (es. N=4)

Organizzazione interallacciata

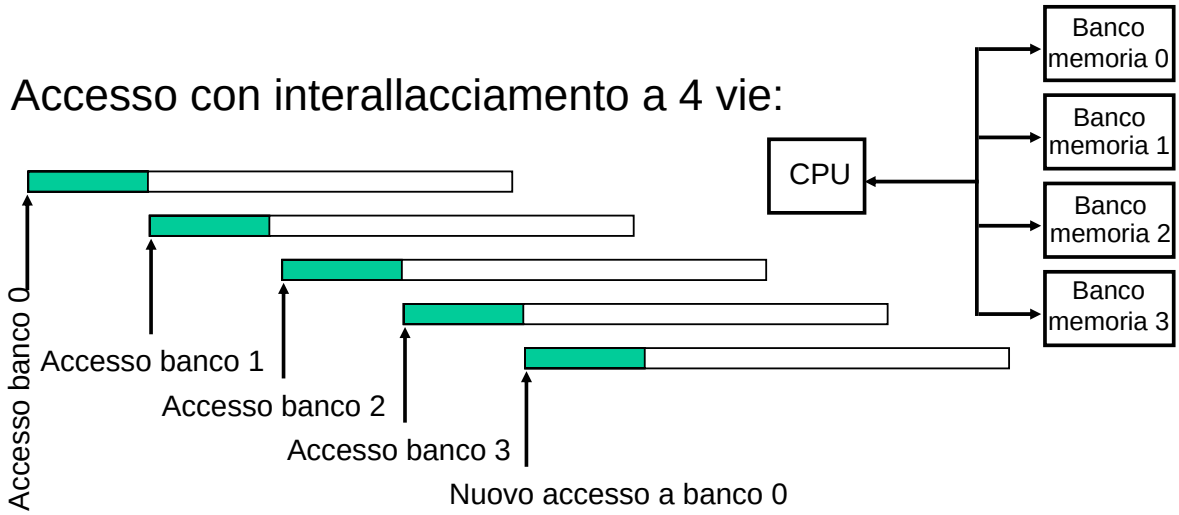
- Organizzazione della memoria principale in banchi con accesso indipendente
  - *m* parole aventi indirizzi consecutivi sono allocate in altrettanti banchi di memoria distinti e consecutivi
- L'accesso al blocco da leggere in memoria è fatto in parallelo su ogni banco
  - Obiettivo: accedere in parallelo a tutte le parole in un blocco di *m* parole consecutive
- Bus di larghezza standard

Organizzazione interallacciata (2)

- Accesso senza interallacciamento:



- Accesso con interallacciamento a 4 vie:



Esempio

- Tempi di accesso al sistema di memoria:
  - 1 ciclo di clock del bus per inviare l'indirizzo
  - 15 cicli di clock del bus per ogni accesso in DRAM
  - 1 ciclo di clock del bus per inviare una parola
- Cache con blocchi da 4 parole

- Con organizzazione semplice:
  - Larghezza del bus e della memoria pari ad 1 parola
  - Miss penalty = 1 + 4·15 + 4·1 = 65 cicli di clock del bus
  - Transfer rate = # byte / cicli di clock = 4·4 / 65 = 0,25 B/ciclo

Esempio (2)

- Con organizzazione larga:
  - Larghezza del bus e della memoria pari a 2 parole
  - Miss penalty = 1 + 2·15 + 2·1 = 33 cicli di clock del bus
  - Transfer rate = # byte / cicli di clock = 4·4 / 33 = 0,48 B/ciclo

  - Larghezza del bus e della memoria pari a 4 parole
  - Miss penalty = 1 + 1·15 + 1·1 = 17 cicli di clock del bus
  - Transfer rate = # byte / cicli di clock = 4·4 / 17 = 0,94 B/ciclo

- Con organizzazione interallacciata:
  - Larghezza del bus e della memoria pari a 1 parola e memoria con 4 banchi interallacciati
  - Miss penalty = 1 + 1·15 + 4·1 = 20 cicli di clock del bus
  - Transfer rate = # byte / cicli di clock = 4·4 / 20 = 0,8 B/ciclo