UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II

Calcolatori Elettronici II

Prof.re Nicola Mazzocca

1.1.12. Progetto di unità di memoria (Cache, MMU, Virtual Cache)

In qualsiasi sistema di elaborazione ogni processo, sia per utilizzare i dati sia per prelevare le istruzioni, ha la necessità di interagire frequentemente con la memoria centrale.
Tale tipo di memoria ha una velocità di accesso di gran lunga inferiore rispetto a quella del processore, rappresentando difatti il collo di bottiglia dell'elaborazione.
Per sfruttare più efficientemente la velocità di calcolo e di conseguenza migliorare le prestazioni complessive di funzionamento, la soluzione ingegneristica in assoluto più adottata è quella di introdurre una memoria più veloce tra processore e RAM, chiamata memoria cache.
Gli enormi vantaggi ottenuti dal suo utilizzo, uniti all'elevato costo, fanno del dimensionamento della memoria cache un elemento di fondamentale importanza nella progettazione dei sistemi di elaborazione, ed è quindi opportuno approfondire ulteriormente l'argomento.
Possiamo dire, che il processo che porta al dimensionamento effettivo della cache è relativamente semplice, ma ben più complessa è la filosofia che ci consente di applicarlo.
Quando si effettua un dimensionamento, infatti, si deve avere un criterio di ottimizzazione secondo il quale poter effettuare delle scelte: nel caso delle cache, il dimensionamento ottimale è difficile da ottenere, perché deve essere fatto rispetto ai programmi che utilizzano le risorse del sistema di elaborazione, ognuno dei quali ha un proprio livello di località.
Inoltre, ogni programma non è rappresentato nella sua interezza ma da una sua istanza, che varia di volta in volta in base ai dati che essa tratta.
Per capire meglio quali sono i problemi che si pongono, è prima necessario capire come sono fatte le memorie cache e quali sono le tecniche che ci consentono di interagire con esse.

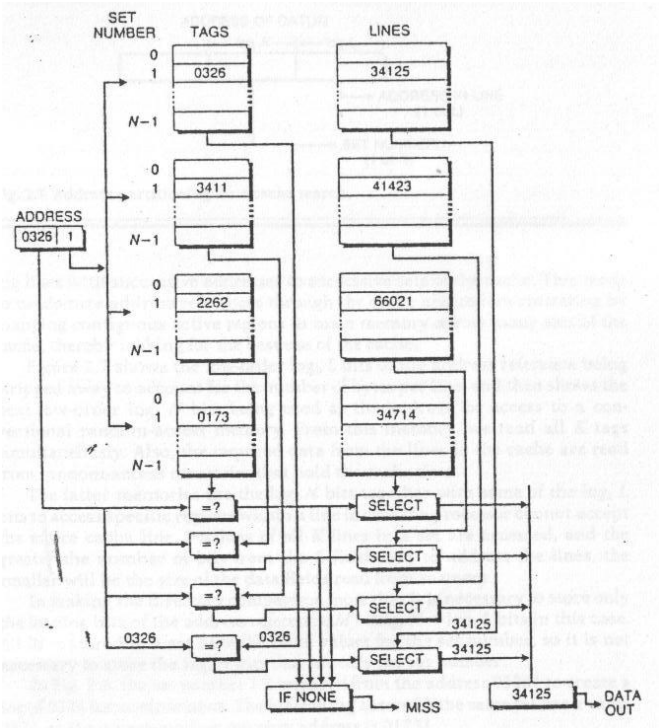


Figura 51: Schema di principio di una cache associativa

La cache è una memoria di tipo associativo. Ciò significa che ogni indirizzo è composto da una chiave, che di fatto identifica un blocco , e da uno spiazzamento che identifica l'elemento all'interno del blocco. Quando si effettua la ricerca di un blocco, la parte di indirizzo che rappresenta la chiave viene confrontata mediante comparatori in parallelo con il campo chiave dei blocchi presenti nella cache, in modo da abilitare eventualmente il blocco cercato portandolo in uscita. Se il dato non è presente, cioè se la chiave non corrisponde a nessun blocco, si genera un cache miss dovuto al fault dei comparatori. Se, viceversa, il blocco è presente (cache hit) si valuta la seconda parte dell'indirizzo (lo spiazzamento all'interno del blocco) che consente di selezionare l'elemento desiderato.

Con la cache il tempo di lettura teoricamente è fissato, poiché la valutazione delle chiavi e degli spiazzamenti di cui si compongono gli indirizzi dei blocchi è effettuata in parallelo: praticamente le due parti in cui è suddiviso ogni indirizzo vengono esaminate contemporaneamente per ogni elemento della cache. Il valore della chiave funge da segnale di abilitazione per uno specifico blocco, di cui la parte descritta dallo spiazzamento viene portata in uscita. Quindi si può pensare di avere anche un bus sincrono per la comunicazione con il processore. Dobbiamo stabilire però come abilitare i segnali di sincronizzazione: di sicuro si necessita di un segnale di Ready che indichi la disponibilità del dato sul bus. Inoltre per poter campionare anche l'eventuale assenza del blocco, cioè il cache miss, è indispensabile un ulteriore segnale (segnale di cache miss) che

consenta alla cache di porre in attesa il processore e contemporaneamente avviare la comunicazione con la RAM per il prelevamento dei dati richiesti. In realtà si avvia il trasferimento in parallelo da RAM e da cache. Se il dato non è presente nella cache allora si genera un cache miss che è un segnale di abilitazione per il dato prelevato dalla RAM. Il rapporto tra cache e RAM così come il rapporto tra cache e processore è tutto realizzato in hw (il processore aspetta in hw ciclando sul flag di ready fino alla risposta della cache). Da quanto detto si intuisce chiaramente che il progetto di una cache è abbastanza complicato, perché essa è assimilabile per metà ad una memoria e per metà ad un processore, in quanto deve comportarsi da memoria verso il processore e da processore verso la memoria centrale.

Dopo questa breve descrizione che rende un po' l'idea sulle problematiche di progettazione, passiamo ad analizzare più nel dettaglio la questione del dimensionamento.
Per dimensionare la cache abbiamo bisogno di stabilire la sua grandezza D (dimensione della cache), ma non solo perché essa, come già detto, non è semplicemente una memoria, ma bensì un' architettura e necessita quindi di ulteriori parametri che la descrivono.

Tali parametri sono: il grado di associatività K (quanti comparatori ha la cache). A parità di memoria il grado di associatività determina la lunghezza dei blocchi. Questo significa che, fissata la dimensione D, una memoria cache con 4 comparatori possiede dei blocchi di lunghezza doppia rispetto a quelli della stessa memoria con 8 comparatori, e 4 volte più grandi di quelli che avrebbe con 16 comparatori.
Da questa considerazione notiamo già che identificare una cache solo attraverso la sua dimensione non è corretto. Oltre al grado di associatività, è importante anche conoscere la lunghezza L, cioè quanto è lungo ogni blocco.
Inoltre c'è da dire che nella pratica, per motivi prestazionali, le memorie cache non sono completamente associative, cioè non è possibile assegnare un blocco della memoria centrale ad uno qualsiasi dei blocchi della cache, ma bensì essi vengono assegnati ad insiemi di blocchi prefissati denominati SET. In questo caso le memorie che si realizzano sono denominate set associative[1].
Per tali memorie l'indirizzo di ogni blocco viene visto come suddiviso in tre parti: la chiave che identifica il blocco, il numero di set che identifica l'insieme di blocchi della cache in cui è possibile memorizzare uno specifico blocco della RAM, e lo spiazzamento all'interno del blocco.

In questi casi un altro parametro che caratterizza la memoria cache è proprio il numero di set N, in cui essa è suddivisa. Infatti, ogni set determina le posizioni in cui è possibile trovare un blocco della memoria RAM qualora esso sia presente in cache.

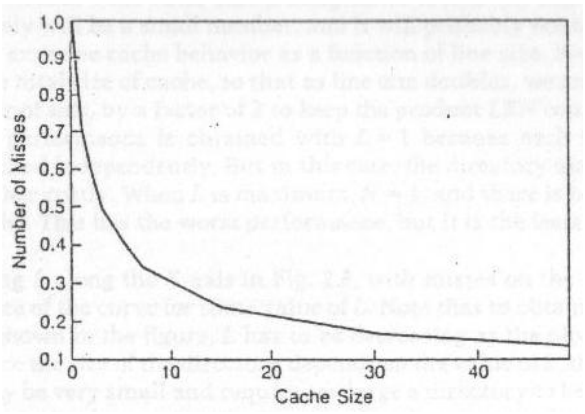


Figura 52: Probabilità di cache miss all'aumentare della grandezza della cache

Da questo capiamo che dimensionare la cache vuol dire scegliere la quantità di memoria che deve possedere. Oltre a ciò (e cosa che ben più difficile) dobbiamo stabilire il *grado di associatività* K, la lunghezza L del blocco ed il numero N di set necessari. La memoria totale è data infatti dalla seguente equazione:

$$D = K * L * N$$

Ricordiamo che la *località dei dati* è favorita con blocchi lunghi, che però si rivelano inutili se in corso di esecuzione del programma si rendono necessari molti salti.

[1] Per un ulteriore approfondimento vedi Hamacker pg.243-249.

Abbiamo dunque 3 gradi di libertà (K, L, N) che possiamo scegliere per progettare la cache, sapendo che la probabilità di cache miss diminuisce all'aumentare della dimensione della cache, e che conviene generalmente sovradimensionare la cache per renderla robusta alle variazioni di carico. Da notare che abbiamo, a fronte di 3 gradi di libertà, una sola equazione disponibile: K * L * N = D. Quest'ultima ha dunque ∞^2 soluzioni. Inoltre ciò che rende complicato il problema è la sua dipendenza dai dati.

Il problema, quindi, risulta matematicamente molto complicato, ma dal punto di vista ingegneristico è più semplice da affrontare se a monte vengono effettuate opportune considerazioni. Difatti, la progettazione è guidata da parametri fisici che limitano il dominio di scelta dei **valori assunti da K, L, N.**

- I tipici *valori assunti da K* sono 4, 8, 16, 32 ma non oltre, perché non ha senso inserire più di 32 comparatori all'interno del sistema di memoria, in quanto non se ne trarrebbe alcun vantaggio.

- *L* è limitata superiormente dal parallelismo del bus utilizzato. Infatti aumentando la lunghezza del blocco, aumenta il numero di operazioni da effettuare per portare il dato in memoria e di conseguenza aumenta il tempo necessario per il trasferimento (le considerazioni fatte valgono per la cache dati, ma non per la cache istruzioni dove viceversa avere blocchi di lunghezza maggiore consente di prelevare più istruzioni contemporaneamente dalla memoria). Altresì valori troppo piccoli rischiano di sovraccaricare eccessivamente l'utilizzo del bus ed aumentano la probabilità di cache miss. *Tipici valori di L sono 64, 128, 256*. In generale si dovrà scegliere un compromesso tra non prendere frequentemente i dati dalla memoria e avere il bus libero. Inoltre bisogna considerare che anche la tecnica di gestione influisce sulla scelta di L. Se si utilizza una tecnica Write Back può essere conveniente avere blocchi di grandi dimensioni, poiché i trasferimenti in memoria sono effettuati in maniera asincrona, cioè indipendentemente dalla frequenza di modifica dei dati

- Infine, il numero di set, una volta fissati D, K ed L, può essere ricavato di conseguenza sfruttando l'equazione D=K*L*N.

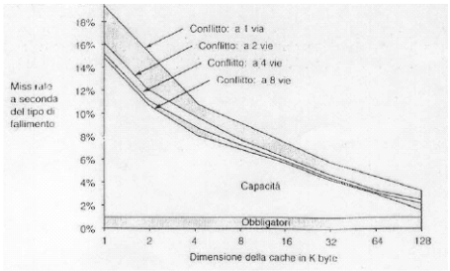
Dimensionamento (1)

- I gradi di libertà nel progetto sono quindi 3:
  - L , k , N ma i parametri hanno dei vincoli correlati tra loro:
- Esistono vincoli tecnologici (K determina il numero di comparatori).
- Esistono vincoli determinati dal principio di località che però dipende fortemente dal programma in esecuzione (carico).
- L viene fissata in funzione della banda di memoria e della politica di write (Wtrough -> Lpiccolo; Wback -> Lgrande)
- Solo per sistemi embedded (carico noto e fissato) posso effettuare delle ottimizzazioni significative.

È lecito a questo punto domandarsi se ha senso utilizzare per le cache una tecnica Write Through.
Tale tecnica infatti prevede che ogni qual volta un dato viene modificato, le modifiche debbano essere trasferite in memoria.
Sembrerebbe, quindi, che tutti i vantaggi scaturiti dall'utilizzo della cache si annullino. In realtà per le sue caratteristiche, la memoria cache può essere vista come una architettura che si comporta da memoria verso il processore e da processore verso la memoria, come abbiamo già osservato prima.
Quindi essa non dialoga direttamente con la RAM, ma lo fa tramite un apposito buffer al quale delega il compito di effettuare i trasferimenti, così da ridurre i tempi di attesa del processore.
Riepilogando dunque, write through e write back non alterano il tempo di scrittura in memoria, ma modificano solo la frequenza di utilizzo del bus.
Inoltre eventuali ottimizzazioni possono essere ottenute solo per sistemi embedded dove sono note a priori le caratteristiche dei programmi ed in particolare la frequenza delle operazioni di modifica dei dati che essi compiono.

Dimensionamento (2)

- Il progetto non prevede un'unica soluzione ma uno schema logico può essere il seguente:
- L'andamento del miss-rate con la dimensione della cache è di tipo a ginocchio dove la posizione di quest'ultimo dipende dal carico e dal processore (in genere raddoppiando la dimensione i miss diminuiscono di circa il 30 %);

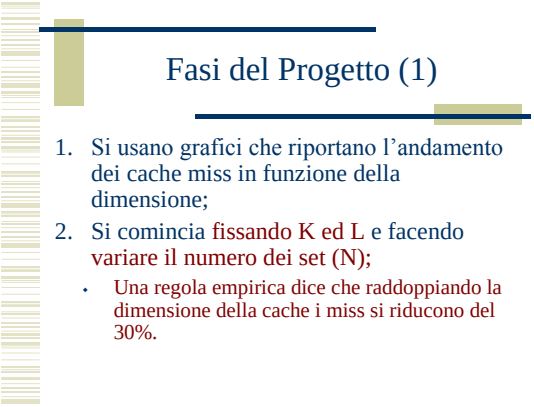


Da quanto detto, non è difficile comprendere che un ruolo di fondamentale importanza è rivestito da una attenta analisi delle applicazioni che utilizzano il sistema.

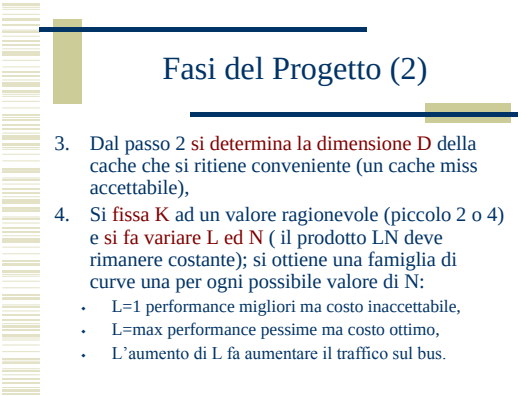
Il problema che in generale si va ad affrontare è che i programmi, così come i dati, hanno una località che risulta estremamente variabile, perché fortemente dipendente dalla particolare istanza di esecuzione.

Ciò implica che, sebbene sappiamo disegnare la forma della curva che indica la variazione di cache miss al variare della dimensione D, è indicato in figura un esempio (decrescente con la cache size), non sappiamo però dove è esattamente posizionata (potrebbe essere più alta o più bassa). Tale posizione è funzione del problema specifico. La cache è progettata sulla base di un carico 'tipico' (insieme tipico di dati trattai dal problema). Il carico tipico lo si ottiene mediante tecniche dette di *clusterizzazione* che approfondiremo in seguito.

Per grandi linee il progetto viene portato avanti partendo da un problema e studiando la sua incidenza sulla cache. Si determina una dimensione ottima per quel particolare problema. Si fa poi lo stesso per un secondo e un terzo problema. Infine si elabora una funzione obiettivo (tipo media pesata) che si propone di minimizzare i costi, si determinano i parametri di una cache che vada bene per tutti e tre i casi e si effettua una simulazione nei tre problemi con quella particolare scelta di parametri per vedere se la cache ottenuta è adeguata.
Il limite principale è dato dal fatto che il grado di associatività varia secondo le potenze di due, con intervallo tipico di variazione $2 < K < 8$. Per il numero di set si ha $8 < N < 32$. Anche la lunghezza del blocco non può essere eccessiva, perché il bus ne risulterebbe sovraccaricato. 'La tecnologia non dà soluzioni, ma limiti'. Fra le ∞^2 soluzioni dobbiamo essere in grado di trovarne una sub-ottima, compatibile coi vincoli tecnologici, non essendo possibile in generale trovare la soluzione migliore in assoluto. All'aumentare del numero di simulazioni aumenta comunque la vicinanza della soluzione trovata a quella ottima in assoluto.
Si usano quindi tecniche prese a prestito dalla Ricerca Operativa, con particolare riferimento al Branch & Bound, che, data una soluzione sub-ottima, è in grado di dire quale soluzione ottima si troverebbe, proseguendo con l'algoritmo in quella direzione, e permette quindi di decidere se conviene proseguire o accontentarsi della soluzione già trovata.

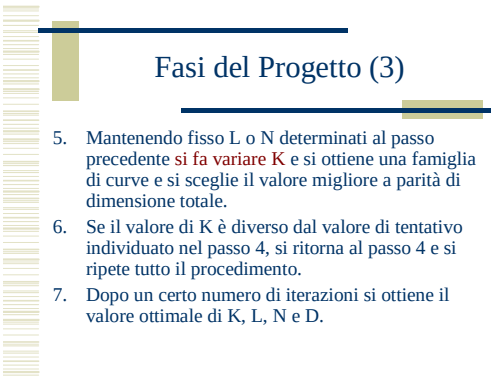


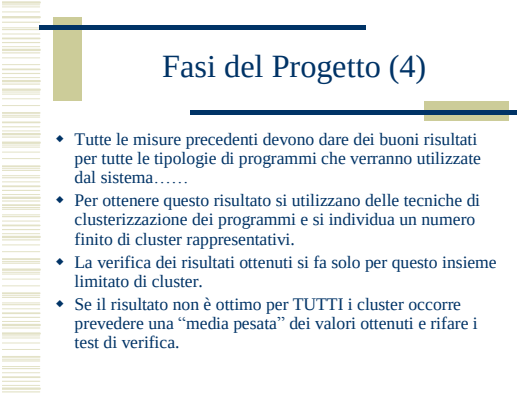
La prima fase del dimensionamento consiste nello stimare D: si fissano i valori di K (es. 4) ed L (256) e facendo variare N si definisce il valore di D. Con una simulazione si vede di volta in volta quanti cache-miss si riscontrano; si costruisce cioè la curva di decrescenza dei cache-miss per quel particolare problema e quei fissati valori K, N. Si sceglie dunque la quantità di memoria D che serve sulla base di una quota di cache miss ritenuta accettabile. Una semplice regola empirica dice che raddoppiando la dimensione della cache i cache-miss diminuiscono grossomodo del 30%. Ovviamente, da un certo punto in poi il guadagno in cache-miss diventa troppo contenuto perché convenga continuare a raddoppiare D.



È logico ora effettuare un'analisi di sensibilità anche per K ed L. C'è da dire comunque che K ed L dipendono in larga misura da fattori tecnologici (rispettivamente, numero di comparatori e banda del bus), per cui si ha

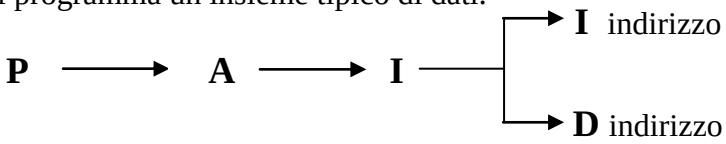
una minore libertà pratica nella loro scelta.





Scelta una dimensione ragionevole D della cache, si fissa dunque K ad un valore piccolo e diverso da 1 (altrimenti avremmo una memoria tradizionale) e si fanno variare i fattori L ed N (il loro prodotto deve rimanere costante, essendo stati fissati D e K). Si rifanno dunque le simulazioni; la curva cambia, e può essere migliore o peggiore. Se è peggiore, vuol dire che i valori di L ed N trovati al passo precedente erano quelli ottimali. Se è migliore, consideriamo i nuovi valori di L ed N. Si noti che per L = 1 abbiamo le performance migliori in assoluto, ma ad un costo inaccettabile, perché il numero di set è troppo elevato, mentre per L massimo i blocchi sono troppo lunghi (performance peggiore a costo minimo).
A questo punto abbiamo una terna (K, L, N) e resta da far variare solo il grado di associatività K (numero di comparatori). Fissati L o N in base al precedente passo si studiano le performance al variare del grado di associatività. Si ottiene una nuova famiglia di curve.
Se il valore di K ottenuto in corrispondenza di prestazioni significativamente migliori è minore o maggiore di quello che era stato fissato in precedenza, si deve ripetere il procedimento dall'inizio e più precisamente dal punto in cui si fissa un K e si fanno variare L e N. Infine, il valore di K trovato sarà quello giusto. Il procedimento viene allora ripetuto per un secondo problema.
Trovato l'ottimo per un certo numero di problemi, poniamo 3, si effettuano delle simulazioni conclusive per verificare l'efficienza con la cache così dimensionata.

Ma come si fa a simulare una cache?
Supponiamo di avere un problema P risolto da un algoritmo A. A compilato diviene un insieme di istruzioni; ogni istruzione può accedere all'indirizzo di un'altra istruzione o di dati. Supponiamo di dare in ingresso al programma un insieme tipico di dati.



È possibile sapere quali indirizzi usa un determinato programma, ovvero ricavare la *traccia* del programma (in ogni istante, in quale indirizzo sta leggendo o scrivendo). È anche necessario per la simulazione conoscere se l'operazione eseguita sui vari dati è di volta in volta di lettura o di scrittura (r/w), mentre non importa sapere quali siano esattamente i dati, ma solo la loro posizione di memoria. La traccia si può avere sia mediante un simulatore software (es. simulatore di processore tipo ASIM, o simulatore che fa l'analisi statica del codice, o ancora simulatore che si limita a fare una trap su ogni istruzione) che nello stesso hardware (monitor hardware). I monitor hardware hanno il vantaggio di essere meno 'intrusivi'. Il simulatore software consente anche di specificare se le memorie cache dati e istruzioni debbano essere unite o separate.
In ogni caso si ottiene una 'traccia' (nel senso che si è indicato), relativa ad un determinato programma, ad esempio di videoscrittura. Indichiamo con M' ed M'' i dati che rispettivamente sono

e non sono presenti all'interno della cache. Supponiamo che la cache sia semplicemente una tabella (ignoriamo cioè la strutturazione in Set). Analizzando la traccia, possiamo fare una prima, rozza analisi con la quale stabiliamo solamente quanti cache miss si hanno con le memorie M' ed M'' delle dimensioni date. Un successivo raffinamento del modello prevede di simulare anche la presenza dei set.
A questo punto, però, è utile domandarsi come rappresentare le tracce (**Rappresentatività delle tracce**). Questo problema può essere suddiviso in tre sottoproblemi:

1. **Rappresentatività del carico**. Il carico usato per generare la traccia è un carico tipico. Questo viene trovato mediante l'utilizzo di opportune tecniche dette di clusterizzazione basate su formule matematiche, che si fondano sul seguente concetto:
   Teoricamente un programma è rappresentabile come un dato (un punto) in uno spazio ad n dimensioni. Quindi una serie di programmi, o di istanze di programmi di una medesima applicazione possono essere rappresentati da una serie di punti in tale spazio che formano uno o più insiemi.
   L' operazione di clusterizzazione consiste nel trovare un punto che si trova ad una distanza media o, secondo una certa metrica accettabile, ad una distanza geometrica minima (non necessariamente quella euclidea) da tutti i punti dell'insieme a cui esso appartiene.
   Il punto trovato individua un programma che è rappresentativo di una classe di programmi,ovvero è una rappresentazione significativa di un determinato carico. (N.B. il punto non necessariamente appartiene a qualche specifica elaborazione effettivamente realizzata nel sistema, ma mediamente ne rappresenta un insieme).
   Ripetendo il procedimento per diverse applicazioni, e quindi per diversi insiemi, è possibile individuare per ognuna di esse una o più tracce di riferimento.
   Tali tracce devono essere utilizzate per la simulazione.
   Ciò comunque non è facile, perché le attività a cui la macchina è destinata possono essere molto diverse (problemi e programmi diversi). Non esistono criteri generali per intervenire su questo punto.
2. **Transitori**. Il transitorio di inizializzazione della cache può falsare il risultato. Nella fase iniziale, la tabella che rappresenta la cache è vuota e quindi abbiamo una sequenza di cache miss. Ad esempio il primo accesso alla cache dà luogo ad un cache miss, che in condizioni reali (senza transitori) può essere invece un hit. Il dimensionamento della cache, effettuato normalmente per il funzionamento a regime, non deve dipendere da tale transitorio. Gli effetti dei transitori sono diversi a parità di traccia per cache di diverse dimensioni: il transitorio cresce al crescere delle dimensioni della cache. Eliminazione dei miss dovuti al transitorio: questi miss possono essere riconosciuti inizializzando la cache con valori illegali. Dal numero di valori illegali ancora presenti in tabella ad un certo punto della simulazione si riesce a capire qual è stata l'incidenza del transitorio.
3. **Accuratezza della misura**. La traccia non deve essere troppo corta per poter fornire una misura accurata. Deve cioè rilevare un numero di miss sufficienti a misurare con accuratezza il miss ratio. Esiste una regola empirica che afferma che la sua lunghezza deve essere proporzionale alla dimensione della cache elevato a 1.5. Questo vuol dire per esempio che quadruplicando la dimensione della cache, la dimensione della traccia deve essere moltiplicata per 8. Se la cache misura 2 MB, si richiedono circa 100 milioni di riferimenti.

Altre tecniche di dimensionamento. Abbiamo già visto una tecnica di dimensionamento cache che prende in esame più parametri in una sola volta. In questo modo si ha il vantaggio di dover ricorrere ad un numero minore di prove.

Facciamo un breve cenno ad altre due tecniche. La prima è detta delle **ANALISI MULTIPLE**. Simulando una cache con grado di associatività K si possono ottenere anche risultati relativi a cache con grado di associatività minore di K. Ciò significa che se facciamo l'analisi per K = 8 abbiamo di riflesso anche i risultati per K = 4 e K = 2. Se ricordiamo che nel caso visto prima, occorreva far

variare K fra diversi valori per avere un buon dimensionamento, riscontriamo che la tecnica delle analisi multiple è sicuramente vantaggiosa. La tecnica però funziona correttamente se la politica di *replacement* è LRU (Last Recent Used) , o comunque rispetta il principio di inclusione (il set di una cache k-associative deve includere i corrispondenti set di una cache di grado di associatività minore).
Consideriamo il caso (a) illustrato in figura. La cache è costituita da un solo set di 8 elementi. Supponiamo che il più recentemente utilizzato, A, si trova in testa, e il meno recentemente utilizzato, H, in coda. Quando arriva Z, A scende di una posizione (shift) e H esce dalla cache, secondo quanto dettato dal paradigma LRU. Si noti ora che se la cache fosse stata suddivisa in due set di 4 elementi ognuno, in seguito a tale shift, D sarebbe uscito dal primo set, e se fosse stata realizzata con set di 2 elementi,sarebbe uscito B dal primo set. Quindi se in seguito dovessimo accedere all'indirizzo D, avremmo nei casi di cache a 2 o 4 set, dei miss, mentre nel caso a 8 elementi (un solo set) un hit. Se dovessimo accedere all'indirizzo B avremmo un miss solo nell' ultimo caso (cache suddivisa in 4 set).

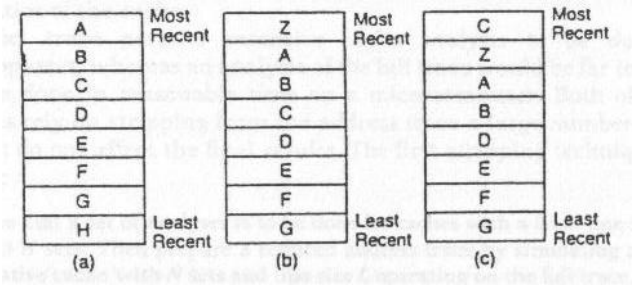


Figura 54: Tecnica delle Analisi Multiple

Se ora C, già presente in cache, dev'essere letto, guadagna la posizione di elemento più recentemente utilizzato e si ha un nuovo scorrimento verso il basso, che però coinvolgerà solo gli elementi fra Z e B (v. figura (c)).
Dal comportamento della cache appena illustrato, si evince chiaramente che, a parità di dimensione, aumentando il grado di associatività, con l' utilizzo della tecnica LRU, il miss rate può solo diminuire.
Altra tecnica: **CAMPIONAMENTO DELLA TRACCIA**. Si estrae una parte della traccia mediante campionamento e la si applica ad uno solo dei set, determinando il comportamento che si avrebbe su di una cache più grande. L'ipotesi (vera solo nel caso in cui i dati siano distribuiti uniformemente su tutti i set) è che il comportamento statistico di tutti i set sia uguale, e quindi si possono fare analisi anche su di un solo set.

1.1.13. Dimensionamento con strumenti simulativi di memorie cache

Per concludere la trattazione delle cache, presentiamo un esempio di dimensionamento mediante l'utilizzo del supporto di simulazione DINERO IV.
Tale simulatore mette a disposizione 3 tracce che sono emblematiche delle tre modalità di funzionamento del processore:

- cpu_bound: Calcolo matematico;
- io_bound: Spice (elaborazione di circuiti elettronici);
- interattivi con l'utente: Word porcessor.

Se si volesse aggiungere una ulteriore traccia, si potrebbe considerare il listato assembler di un programma per determinarne gli indirizzi di accesso in memoria.
Esempio:
Si dimensioni una memoria cache Set Associative, utilizzando DINERO IV, per la quale sono simulate 3 tracce:

- cc1.din : che è il tracciato di un programma di calcolo;
- spice.din : che è tracciato di una simulazione spice;
- tex.din : che è tracciato di un'elaborazione tex.

Le specifiche del progetto sono:

- Cache unica per dati e istruzioni.
- Protocollo Write Back
- Protocollo Write Allocate
- Politica di sostituzione LRU

- 1% < Cache Miss < 3%
- 60% cc1.din;
- 10% tex.din
- 30% spice.din

Si sceglie K non molto alto per non complicare la realizzazione circuitale della memoria: **K=2** e **K=4;** per evitare di sprecare risorse i blocchi hanno lunghezza limitata: **8 < L < 32.**

D	K	L	N	CC1.DIN (%)	TEX.DIN (%)	SPICE.DIN (%)	MEDIA PESATA (%)
8K	2	8	512	12,1	0,92	5,01	8,855
8K	2	16	256	8,2	0,49	3,62	6,055
8K	2	32	128	5,97	0,27	2,79	4,446
8K	4	8	256	10,79	0,92	3,74	7.688
8K	4	16	128	7,3	0,46	2,77	5,257
8K	4	32	64	5,32	0,23	1,92	3,791

Le simulazioni effettuate per D = 8KB mostrano che per nessuna combinazione di L, K e N è possibile soddisfare le specifiche assegnate.
Si procede, dunque, simulando memorie cache per D = 8/16/32/64 KB e si ottengono i seguenti valori:

D	K	L	N	CC1.DIN (%)	TEX.DIN (%)	SPICE.DIN (%)	MEDIA PESATA (%)
16K	2	8	1024	6,82	0,7	2,15	4,807
16K	2	16	512	4,74	0,35	1,52	3,335
16K	2	32	256	3,6	0,18	1,07	2,499
16K	4	8	512	5,57	0,66	1,32	3.804
16K	4	16	256	3,87	0,34	0,91	2,629
16K	4	32	128	2,93	0,17	0,66	1,973
32K	2	8	2048	4,03	0,59	1,13	2,816
32K	2	16	1024	2,61	0,3	0,69	1,803
32K	2	32	512	1,86	0,15	0,47	1,272
32K	4	8	1024	3,35	0,58	0,91	2,341
32K	4	16	512	2,15	0,29	0,54	1,481
32K	4	32	256	1,48	0,15	0,33	1,002
64K	2	8	4096	2,97	0,59	0,76	2,069
64K	2	16	2048	1,8	0,29	1,43	1,538
64K	2	32	1024	1,14	0,15	0,25	0,774
64K	4	8	2048	2,75	0,58	0,72	1,924
64K	4	16	1024	1,61	0,29	0,4	1,115
64K	4	32	512	0,99	0,15	0,22	0,675

Le simulazioni che riescono a soddisfare le specifiche sono le seguenti:

D	K	L	N	CC1.DIN (%)	TEX.DIN (%)	SPICE.DIN (%)	MEDIA PESATA (%)
16K	2	32	256	3,6	0,18	1,07	2,499
16K	4	16	256	3,87	0,34	0,91	2,629
16K	4	32	128	2,93	0,17	0,66	1,973
32K	2	8	2048	4,03	0,59	1,13	2,816
32K	2	16	1024	2,61	0,3	0,69	1,803
32K	2	32	512	1,86	0,15	0,47	1,272
32K	4	8	1024	3,35	0,58	0,91	2,341
32K	4	16	512	2,15	0,29	0,54	1,481
32K	4	32	256	1,48	0,15	0,33	1,002
64K	2	8	4096	2,97	0,59	0,76	2,069
64K	2	16	2048	1,8	0,29	1,43	1,538
64K	4	8	2048	2,75	0,58	0,72	1,924
64K	4	16	1024	1,61	0,29	0,4	1,115

Un buon compromesso tra dimensione, grado di associatività e percentuale di Cache Miss è una memoria da 32 KB, con grado di associatività K=2, la lunghezza del blocco L 32 Byte, numero di set N=512.
L'esempio ci fa capire come funziona il procedimento. Una volta fissato D e trovati K,L,N in maniera approssimata,bisogna procedere con una simulazione, verificando che la percentuale di cache miss imposta sia rispettata. Se ciò non si verifica, allora bisogna scegliere un valore di D diverso e ripetere la simulazione. Ovviamente, si capisce che questo è un processo iterativo, dato dal fatto che non è possibile usufruire di nessuna formula matematica. La difficoltà insita risiede proprio nella simulazione, la quale produce risultati che devono essere opportunamente interpretati attraverso parametri fisici. Una volta fissato D, si determinano i valori di K ed L in base a vincoli tecnologici i quali consentono di restringere il numero di combinazioni ottenibili. Di conseguenza, infine, si ricava N.

Tale modo di procedere implica che debbano eseguirsi molte simulazioni, per ogni tipologia di applicazione. A questo proposito, però, possiamo dire che esistono delle tecniche matematiche inventate da studiosi Giapponesi che servono ad ottimizzare il numero di simulazioni che si possono fare, mediante delle soluzioni che si chiamano dominanti.
Infine, però, bisogna ribadire che il problema del dimensionamento è in generale molto complicato da risolvere, poiché rappresenta un problema mal posto che non è risolvibile matematicamente, ma solo attraverso simulazioni che individuano soluzioni sub-ottime. Inoltre, le simulazioni stesse non possono essere effettuate per tutti i possibili programmi, ma solo su una piccola parte che deve essere quanto più rappresentativa possibile.
In più, bisogna considerare che non ha senso cercare a tutti i costi la soluzione ottima quando si hanno così tanti gradi di aleatorietà.
L'unica certezza matematica che si ha è quella che aumentando la dimensione della cache, diminuisce la probabilità di cache miss.