La memoria cache
Miglioramento delle prestazioni

Testo di riferimento : Hennessy & Patterson: Computer architecture, a quantitative approach (Morgan Kaufmann eds.)

Cache Performance

CPU time = (CPU execution clock cycles + Memory stall clock cycles) x clock cycle time

Memory stall clock cycles = (Reads x Read miss rate x Read miss penalty + Writes x Write miss rate x Write miss penalty)

Memory stall clock cycles = Memory accesses x Miss rate x Miss penalty

Cache Performance

CPUtime = IC x ($CPI_{execution}$ + Mem accesses per instruction x Miss rate x Miss penalty) x Clock cycle time

Misses per instruction = Memory accesses per instruction x Miss rate

CPUtime = IC x ($CPI_{execution}$ + Misses per instruction x Miss penalty) x Clock cycle time

Dinero IV

Dinero è un cache simulator che dato un trace di riferimenti in memoria produce in seguito alla simulazione della cache statistiche sulla cache

Per eseguire dinero è necessario lanciare dineroIV.exe

dineroIV.exe -help fornisce tutte le possibili opzioni

Principali opzioni di dinero

Dinero può essere lanciato dal front end di edumips

Le principali opzioni per la configurazione della cache da simulare

-lN-Tsize P Size

-lN-Tbsize P Block size

-lN-Tassoc U Associativity (default 1)

N livello della cache {1, 2 ,.. 5 }

T tipo di cache {u=unificata, d=data, i=instruction}

U unsigned integer

P unsigned integer con suffisso optionale [kKmMgG]

Esempio Calcolo CPU time mediante dinero

Considerando una frequenza della CPU f = 125 MHz,

calcolare CPI e CPUtime (comprensivo di unità di misura) nei casi seguenti.

a) memoria cache unica senza stalli (caso ideale);

b) cache L1 unificata di 2 K, blocco 32 byte, associativita' 1, miss penalty 80 cicli;

c) cache istruzioni di 1 KByte, blocco 16 byte, associativita' 4, miss penalty 40 cicli;
cache dati di 1 Kbyte, blocco 8 byte, associativita' 4, miss penalty 30 cicli.

Clock Cycle Time= 1/f= 8 ns

Esempio Calcolo CPU time mediante dinero

Lanciamo il sorgente.s mediante edumpis ottenendo al termine dell'esecuzione i seguenti risultati

209 cicli, 174 istruzioni e CPI_{EDU}=1,201

CPI_{EDU}= 1+Data hazard stall per instruction+Control hazard per instruction

a) nel caso ideale misses per instructions = 0

CPUtime= IC * CPIexe*T

CPIexe= CPI_{EDU}+ Structrural Hazard Stall per instruction

Structrural Hazard Stall per instruction= fStruct Haz*1= $f_{LD/SD}$*1

$f_{LD/SD}$ non si può vedere da EDUMPIS. Si ricava da dinero considerando il rapporto tra dati e riferimenti alle istruzioni

Esempio Calcolo CPU time mediante dinero

Lanciando dinero con i parametri di default otteniamo

Metrics	Total	Instrn	Data	Read	Write	Misc
Demand Fetches	275	189	86	50	36	0
Fraction of total	1.0000	0.6873	0.3127	0.1818	0.1309	0.0000

Il numero di accessi ai dati è 86

$f_{LD/SD}$ = data/IC =86/174= 0,494

$CPIexe=CPI_{EDU}+ f_{LD/SD} * 1=1,201+0,494=1,695$

CPUtime= IC * CPIexe*T= 174*1,695*8 ns= 2359,44 ns

Esempio Calcolo CPU time mediante dinero

b) cache L1 unificata di 2 K, blocco 32 byte, associativita' 1, miss penalty 80 cicli;

Lanciamo dinero con i seguenti parametri di linea

-l1-usize 2k -l1-ubsize 32

Otteniamo

Metrics	Total	Instrn	Data	Read	Write	Misc
Demand Fetches	275	189	86	50	36	0
Fraction of total	1.0000	0.6873	0.3127	0.1818	0.1309	0.0000
Demand Misses	11	5	6	2	4	0
Demand miss rate	0.0400	0.0265	0.0698	0.0400	0.1111	0.0000

Misses $_{Overhead}$ = Misses per instruction* Miss penalty = 11/ 189 * 80 = 4,656

CPUtime= IC * (CPIexe+Misses $_{Overhead}$) *T

Esempio Calcolo CPU time mediante dinero

Poiché la cache è unica come nel caso a), CPIexe è uguale a quella della cache ideale

CPIexe=1,695

CPU_{time}= IC * (CPI_{exe}+Misses $_{Overhead}$) T = 174*(1,695+4,656)*8 ns=8840,59 ns

c) cache istruzioni di 1 KByte, blocco 16 byte, associativita' 4, miss penalty 40 cicli;
cache dati di 1 Kbyte, blocco 8 byte, associativita' 4, miss penalty 30 cicli.

Lanciamo dinero con i seguenti parametri di linea

-l1-isize 1k -l1-ibsize 16 -l1-iassoc 4 -l1-dsize 1k -l1-dbsize 8 -l1-dassoc 4

otteniamo

Esempio Calcolo CPU time mediante dinero

I1-icache						
Metrics	Total	Instrn	Data	Read	Write	Misc
Demand Fetches	189	189	0	0	0	0
Fraction of total	1.0000	1.0000	0.0000	0.0000	0.0000	0.0000
Demand Misses	10	10	0	0	0	0
Demand miss rate	0.0529	0.0529	0.0000	0.0000	0.0000	0.0000

I1-dcache						
Metrics	Total	Instrn	Data	Read	Write	Misc
Demand Fetches	86	0	86	50	36	0
Fraction of total	1.0000	0.0000	1.0000	0.5814	0.4186	0.0000
Demand Misses	16	0	16	4	12	0
Demand miss rate	0.1860	0.0000	0.1860	0.0800	0.3333	0.0000

Poiché le cache sono separate, $CPIexe=CPI_{EDU}=1,201$

$CPUtime= IC * (CPIexe+Instr\ Misses_{Overhead}+ Data\ Misses_{Overhead}) *T$

$Instr\ Misses_{Overhead}$=I Misses per instr* IMiss penalty = 10/ 189 * 40 = 2,17

$Data\ Misses_{Overhead}$=D Misses per instr* DMiss penalty = 16/ 189 * 30 = 2,54

CPUtime= 174*(1,201+2,17+2,54)*8 ns= 8228,11 ns

Improving Cache Performance

Average memory-access time = Hit time +
Miss rate x Miss penalty (ns or clocks)

- Improve performance by:
  1. Reduce the miss rate,
  2. Reduce the miss penalty, or
  3. Reduce the time to hit in the cache

Reducing Misses

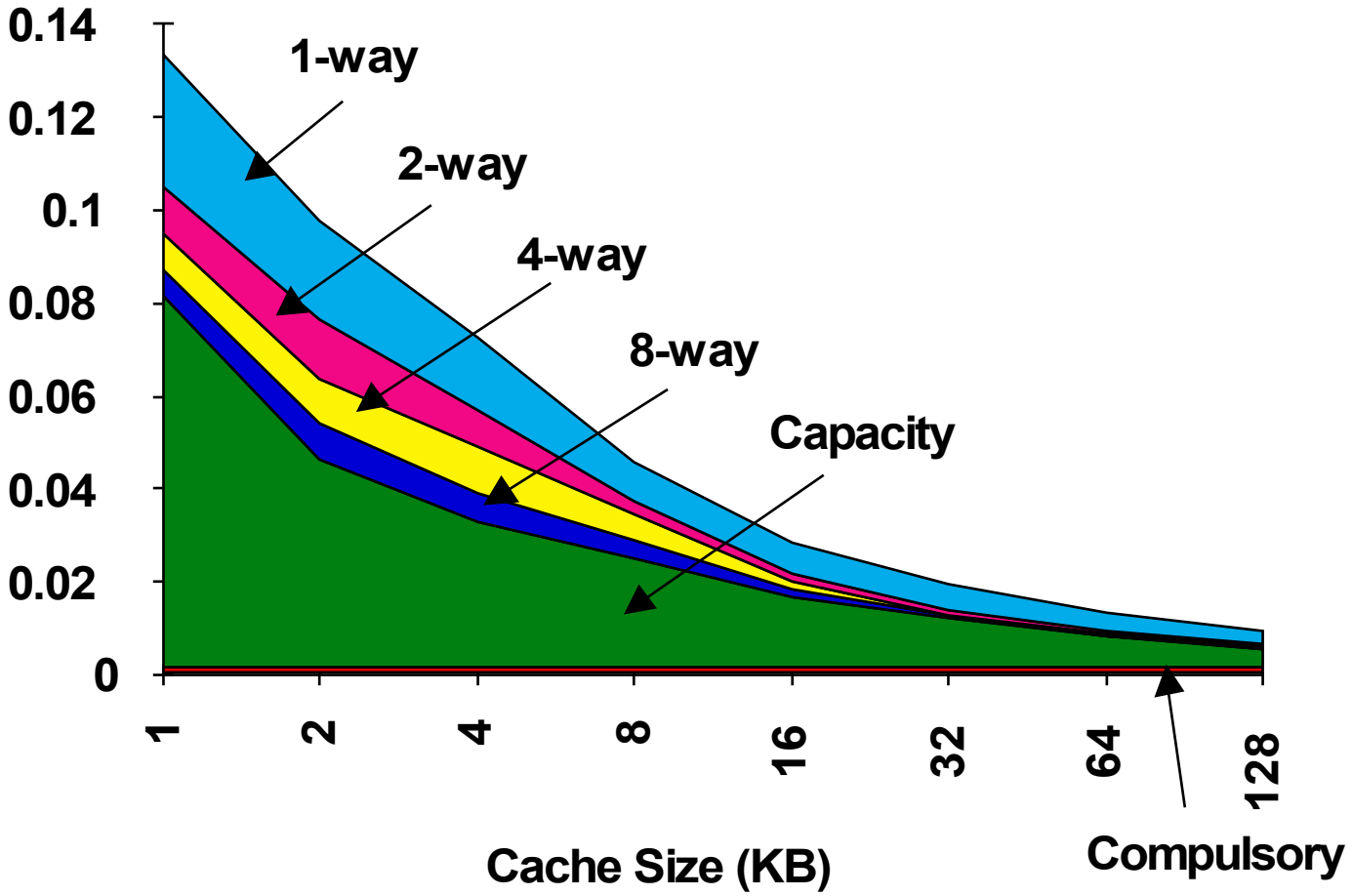
- **Classifying Misses: 3 Cs**

– ***Compulsory***—The first access to a block is not in the cache, so the block must be brought into the cache. These are also called *cold start misses* or *first reference misses*. (Misses in Infinite Cache)

– ***Capacity***—If the cache cannot contain all the blocks needed during execution of a program, capacity misses will occur due to blocks being discarded and later retrieved. (Misses in Size X Cache)

– ***Conflict***—If the block-placement strategy is set associative or direct mapped, conflict misses (in addition to compulsory and capacity misses) will occur because a block can be discarded and later retrieved if too many blocks map to its set. These are also called *collision misses* or *interference misses*.(Misses in N-way Associative, Size X Cache)

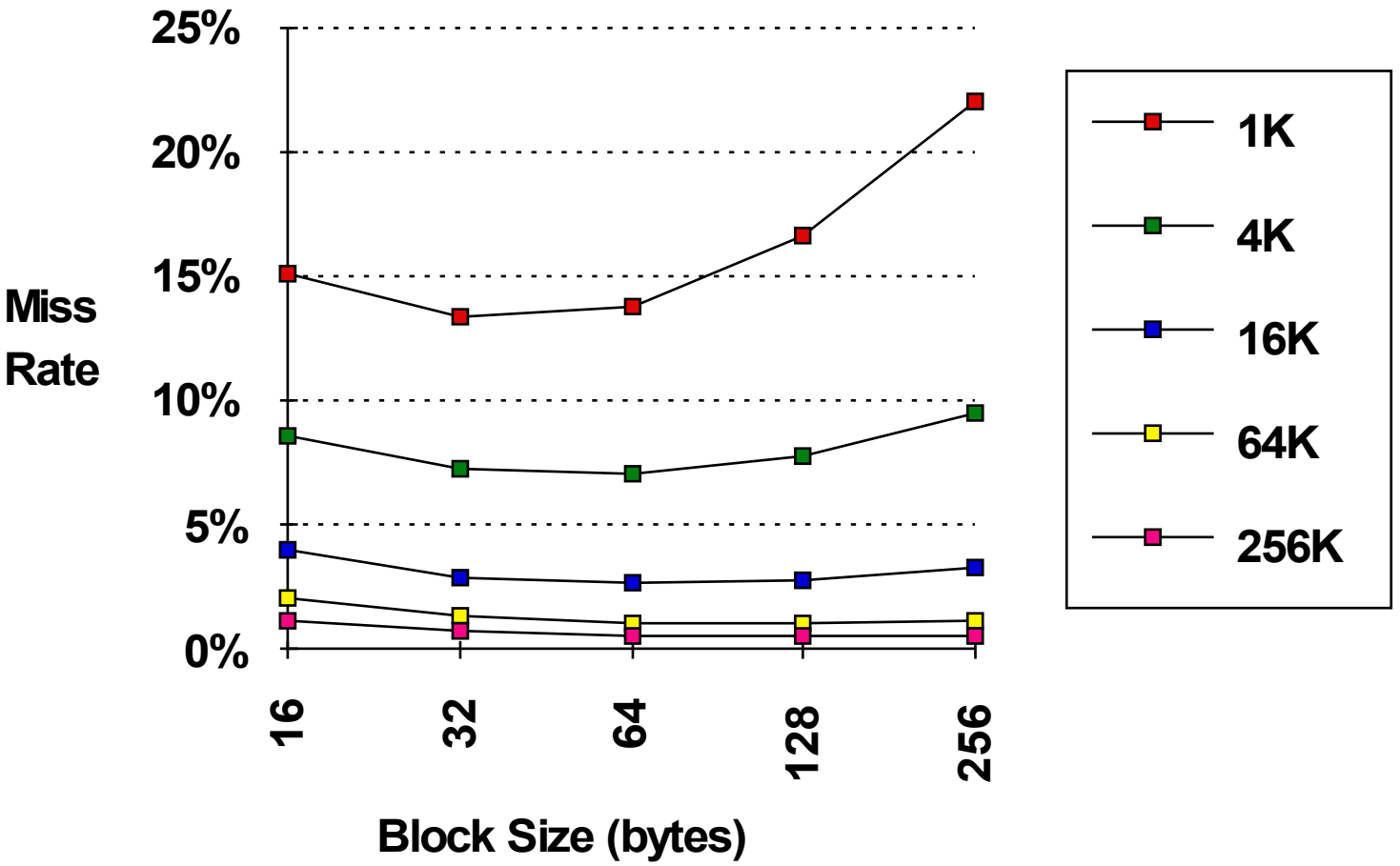
Cause Miss Rate



How Can Reduce Misses?

- Change Block Size? Which of 3Cs affected?
- Change Associativity? Which of 3Cs affected?
- Change Compiler? Which of 3Cs affected?

DIMENSIONE DEL BLOCCO



Dimensione cache e associatività

Miss rate vs A.M.A.T

Cache Size (KB)	Associativity 1-way	2-way	4-way	8-way
1	2.33	2.15	2.07	2.01
2	1.98	1.86	1.76	1.68
4	1.72	1.67	1.61	1.53
8	1.46	1.48	1.47	1.43
16	1.29	1.32	1.32	1.32
32	1.20	1.24	1.25	1.27
64	1.14	1.20	1.21	1.23
128	1.10	1.17	1.18	1.20

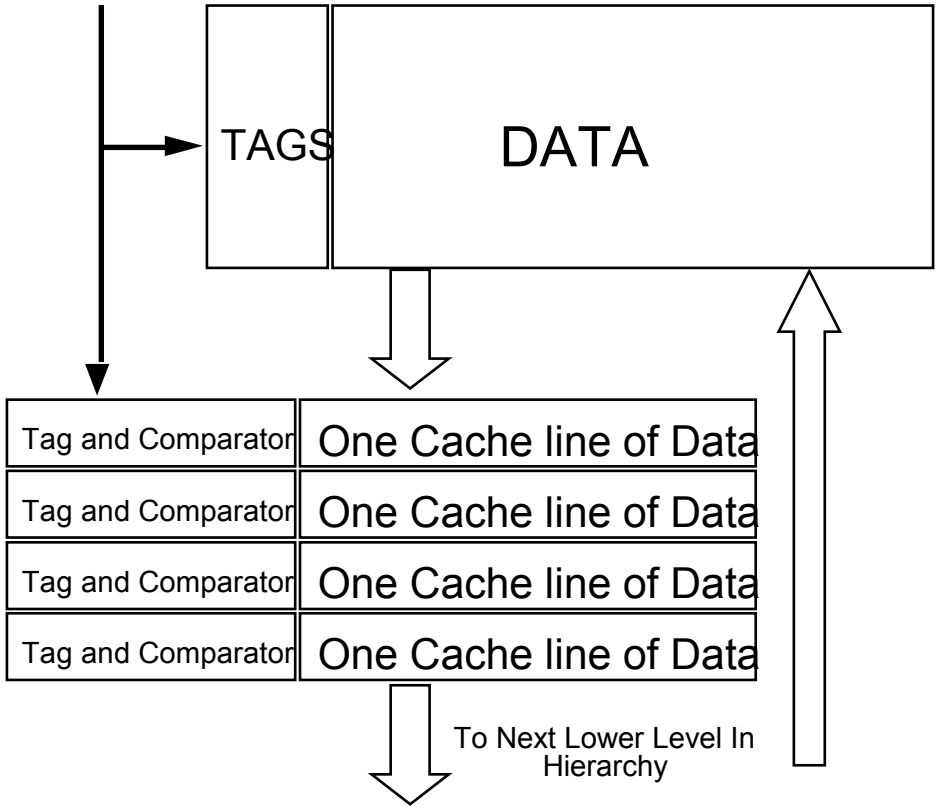
Ridurre il miss rate aumentando l'associatività non necessariamente riduce l'AMAT

Ridurre il Miss Rate mediante Victim Cache

Come combinare un hit time come nella direct mapped riducendo i conflict misses?

Si aggiunge un buffer dove piazzare i dati scartati dalla cache

Jouppi [1990]: Una victim cache di 4 blocchi rimuove dal 20% al 95% dei conflitti per una direct mapped data cache di 4 KB.



Cache Pseudo-associativa

Come combinare il veloce hit time della Direct Mapped con i ridotti conflict misses della 2-way Set Associative cache?

La cache è divisa in due metà.

La verifica dell'hit viene fatta in una metà come una direct-mapped. Nel caso di miss viene testata l'altra metà.

Nel caso di hit nella seconda metà sia ha uno pseudo-hit (hit lento)

Problemi: CPU pipeline è complicato poiché l'hit impiega 1 o 2 cicli

Ideale per cache non collegate direttamente al processore

Reducing Misses by Compiler Optimizations

- **Instructions**

– Reorder procedures in memory so as to reduce misses
– Profiling to look at conflicts
– McFarling [1989] reduced caches misses by 75% on 8KB direct mapped cache with 4 byte blocks

- **Data**

– *Merging Arrays*: improve spatial locality by single array of compound elements vs. 2 arrays
– *Loop Interchange*: change nesting of loops to access data in order stored in memory
– *Loop Fusion*: Combine 2 independent loops that have same looping and some variables overlap

Merging Arrays Example

```
/* Before */
int Val[SIZE];
int Key[SIZE];

/* After */
struct merge {
int Val;
int Key;
};
struct merge merged_array[SIZE];
```

Reducing conflicts between val & key

Loop Interchange Example

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

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

Sequential accesses Instead of striding through memory every 100 words

Loop Fusion Example

```
/* Before */
for (i = 0; i < N; i = i+1)
   for (j = 0; j < N; j = j+1)
      a[i][j] = 1/b[i][j] * c[i][j];
for (i = 0; i < N; i = i+1)
   for (j = 0; j < N; j = j+1)
      d[i][j] = a[i][j] + c[i][j];

/* After */
for (i = 0; i < N; i = i+1)
   for (j = 0; j < N; j = j+1)
      { a[i][j] = 1/b[i][j] * c[i][j];
        d[i][j] = a[i][j] + c[i][j];}
```

2 misses per access to a & c vs. one miss per access

Riduzione del miss penalty

- **Subblock placement**

- **Second Level Cache**

Subblock placement

Non deve essere sostituito un intero blocco su un miss

È presente un bit di validità per ogni sottoblocco

Originariamente inventato per ridurre l'occupazione di memoria per il tag

Tag	validity bit	subblock					
201	1		1		1		1	
302	1		1		0		1	
202	0		0		1		1	
204	0		1		1		1	

Cache di secondo livello

$AMAT = Hit\ Time_{L1} + Miss\ Rate_{L1} \times Miss\ Penalty_{L1}$

$Miss\ Penalty_{L1} = Hit\ Time_{L2} + Miss\ Rate_{L2} \times Miss\ Penalty_{L2}$

$AMAT = Hit\ Time_{L1} + Miss\ Rate_{L1} \times (Hit\ Time_{L2} + Miss\ Rate_{L2} \times Miss\ Penalty_{L2})$

Definizioni:

- *Local miss rate*— miss in questa cache diviso il numero di accessi a questa cache ($Miss\ rate_{L2}$)
- *Global miss rate*—miss in questa cache diviso il numero totale di accessi generati dalla CPU ($Miss\ Rate_{L1} \times Miss\ Rate_{L2}$)

Confronto tra local e global miss rate

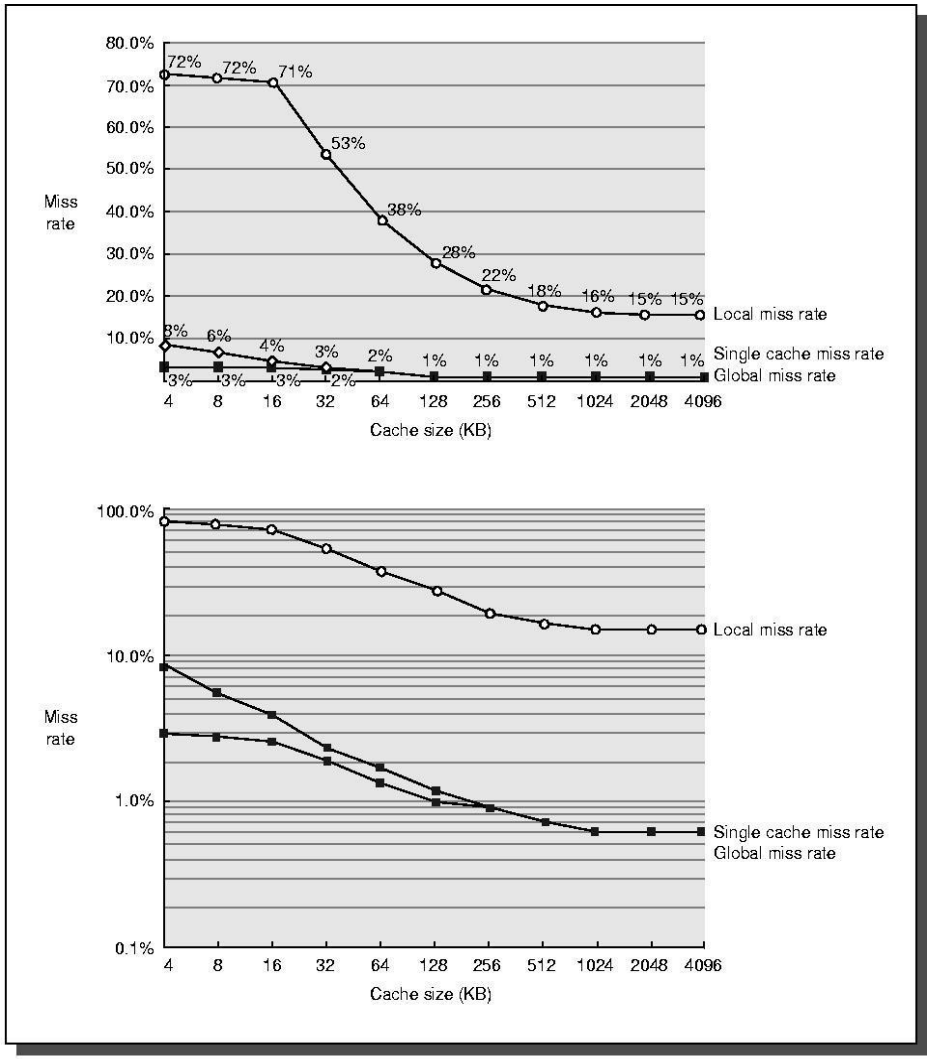
1° livello 32 KByte;
2° livello crescente

Il global miss rate è vicino a quello di una cache con un solo livello se L2 >> L1

Non bisogna usare il local miss rate

Si hanno veloci hit time e un numero inferiore di miss

Poiché gli hit (di L2) sono pochi, l'obiettivo è la riduzione del miss



Main Memory Performance Memory interleaved

Simple:

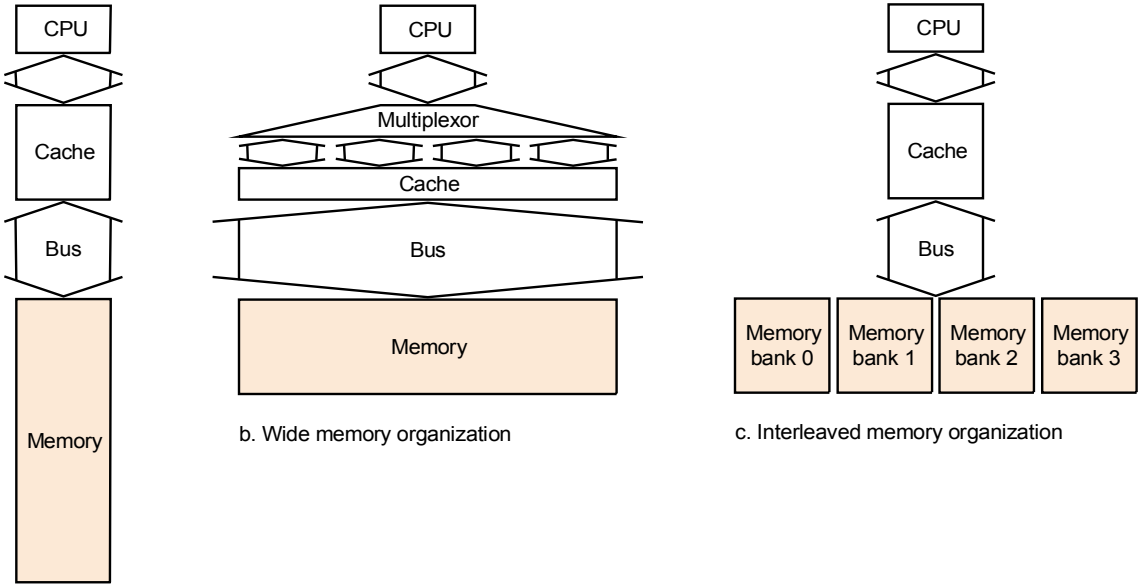
CPU, Cache, Bus, Memory same width (32 or 64 bits)

Wide:

CPU/Mux 1 word; Mux/Cache, Bus, Memory N words (Alpha: 64 bits & 256 bits; UtraSPARC 512)

Interleaved:

CPU, Cache, Bus 1 word: Memory N Modules (4 Modules); example is *word interleaved*



a. One-word-wide memory organization

b. Wide memory organization

c. Interleaved memory organization

Memory Interleaved

Timing model (word size is 32 bits)

1 to send address,

6 access time, 1 to send data

Cache Block is 4 words

Simple M.P. = 4 x (1+6+1) = 32
Wide M.P. = 1 + 6 + 1 = 8
Interleaved M.P. = 1 + 6 + 4x1 = 11