CMSC 611: Advanced Computer Architecture

Cache 2

Some material adapted from Mohamed Younis, UMBC CMSC 611 Spr 2003 course slides
Some material adapted from Hennessy & Patterson / © 2003 Elsevier Science

Measuring Cache Performance

- To enhance cache performance, one can:
  - Reduce the miss rate (e.g. diminishing blocks collisions)
  - Reduce the miss penalty (e.g. adding multi-level caching)
  - Enhance hit access time (e.g. simple and small cache)

$$\text{CPU time} = (\text{CPU execution cycles} + \text{Memory stall cycles}) \times \text{Cycle time}$$

$$\text{Memory stall cycles} = \text{Read stall cycles} + \text{Write stall cycles}$$

$$\text{Read stall cycles} = \frac{\text{Reads}}{\text{Program}} \times \text{Read miss rate} \times \text{Read miss penalty}$$

For write-through scheme:

Hard to control, assume enough buffer size

$$\text{Write stall cycles} = \left(\frac{\text{Writes}}{\text{Program}} \times \text{Write miss rate} \times \text{Write miss penalty}\right) + \text{Write buffer stalls}$$

Example

Assume an instruction cache miss rate for gcc of 2% and a data cache miss rate of 4%. If a machine has a CPI of 2 without any memory stalls and the miss penalty is 40 cycles for all misses, determine how much faster a machine would run with a perfect cache that never missed. Assume 36% combined frequencies for load and store instructions

Answer:

Assume number of instructions = I

Instruction miss cycles = $I \times 2\% \times 40 = 0.8 \times I$

Data miss cycles = $I \times 36\% \times 4\% \times 40 = 0.56 \times I$

Total number of memory-stall cycles = $0.8\,I + 0.56\,I = 1.36\,I$

The CPI with memory stalls = $2 + 1.36 = 3.36$

$$\frac{\text{CPU time with stalls}}{\text{CPU time with perfect cache}} = \frac{I \times CPI_{stall} \times \text{Clock cycle}}{I \times CPI_{perfect} \times \text{Clock cycle}} = \frac{CPI_{stall}}{CPI_{perfect}} = \frac{3.36}{2}$$

What happens if the CPU gets faster?

Classifying Cache Misses

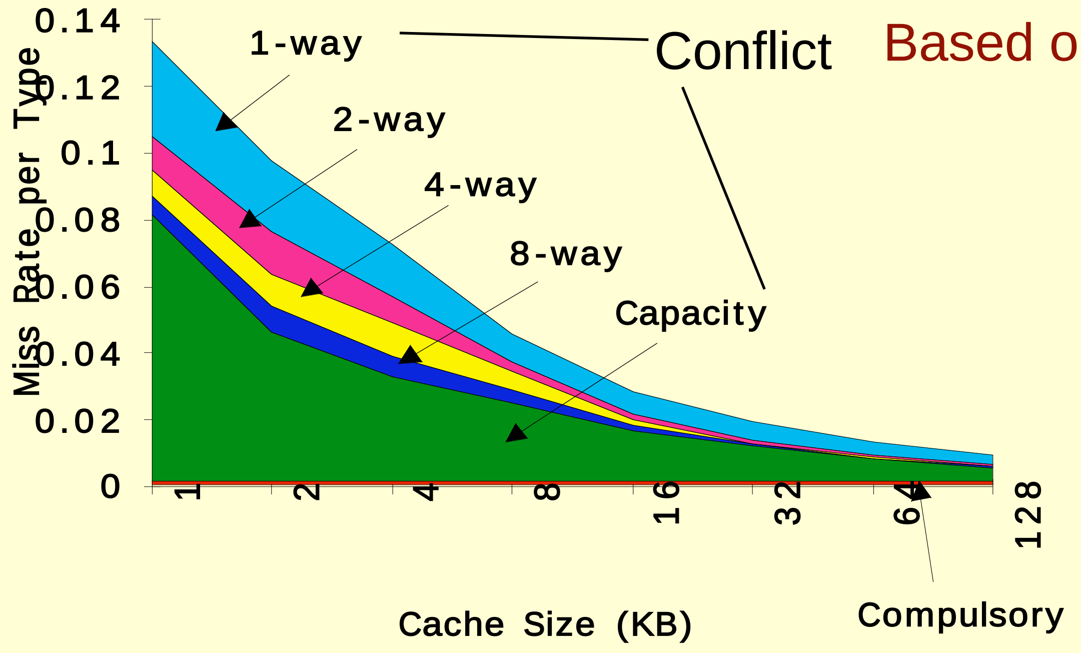
- Compulsory
  - First access to a block not in cache
  - Also called cold start or first reference misses
  - (Misses in even an Infinite Cache)
- Capacity
  - If the cache cannot contain all needed blocks
  - Due to blocks discarded and re-retrieved
  - (Misses in Fully Associative Cache)
- Conflict
  - Set associative or direct mapped: too many blocks in set
  - Also called collision or interference
  - (Misses in N-way Associative Cache)

Improving Cache Performance

- Capacity misses can be damaging to the performance (excessive main memory access)
- Increasing associativity, cache size and block width can reduces misses
- Changing cache size affects both capacity and conflict misses since it spreads out references to more blocks
- Some optimization techniques that reduces miss rate also increases hit access time

Miss Rate Distribution

- Compulsory misses are small compared to other categories
- Capacity misses diminish with increased cache size
- Increasing associativity limits the placement conflicts



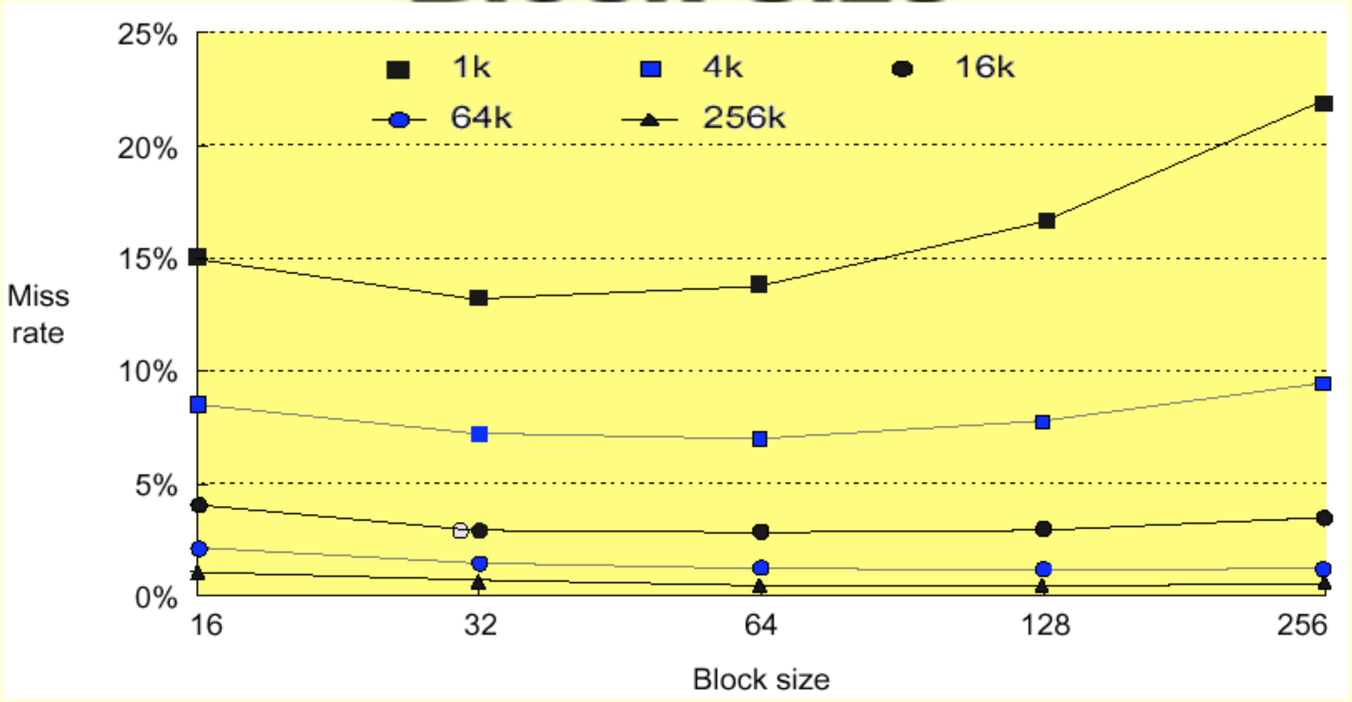
Based on SPEC92

Techniques for Reducing Misses

$$CPUtime = IC \times \left(CPI_{Execution} + \frac{Memory\ \ accesses}{Instruction} \times \boxed{\mathbf{Miss\ rate}} \times Miss\ \ penalty \right) \times Clock\ \ cycle\ \ time$$

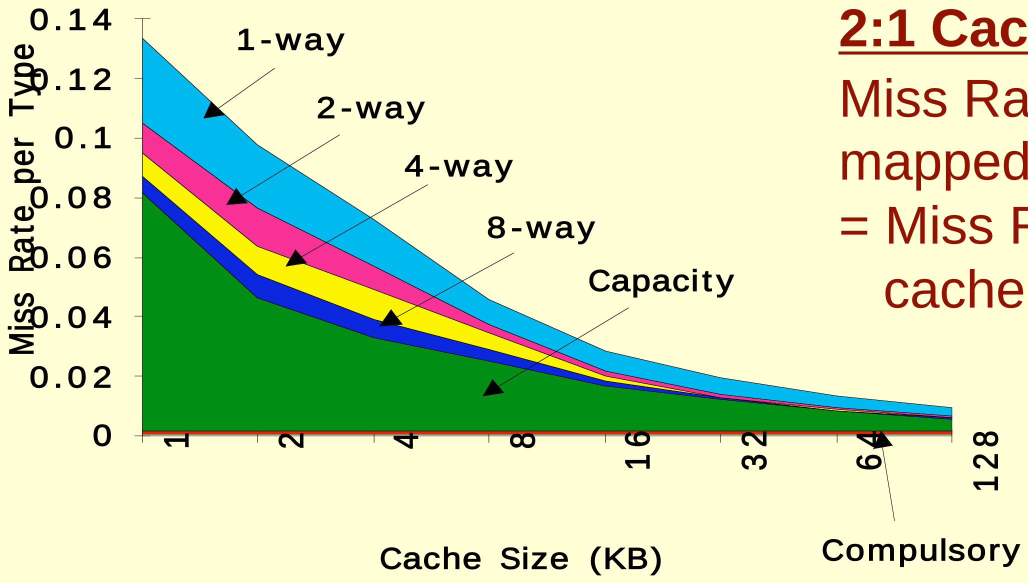
1. Reducing Misses via Larger Block Size
2. Reducing Misses via Higher Associativity
3. Reducing Misses via Victim Cache
4. Reducing Misses via Pseudo-Associativity
5. Reducing Misses by H/W Prefetching Instr. and Data
6. Reducing Misses by S/W Prefetching Data
7. Reducing Misses by Compiler Optimizations

Reduce Misses via Larger Block Size



- Larger block sizes reduces compulsory misses (principle of spatial locality)
- Conflict misses increase for larger block sizes since cache has fewer blocks
- The miss penalty usually outweighs the decrease of the miss rate making large block sizes less favored

Reduce Misses via Higher Associativity



2:1 Cache Rule:

Miss Rate for direct mapped cache of size N = Miss Rate 2-way cache size N/2

- Greater associativity comes at the expense of larger hit access time
- Hardware complexity grows for high associativity and clock cycle increases

Example

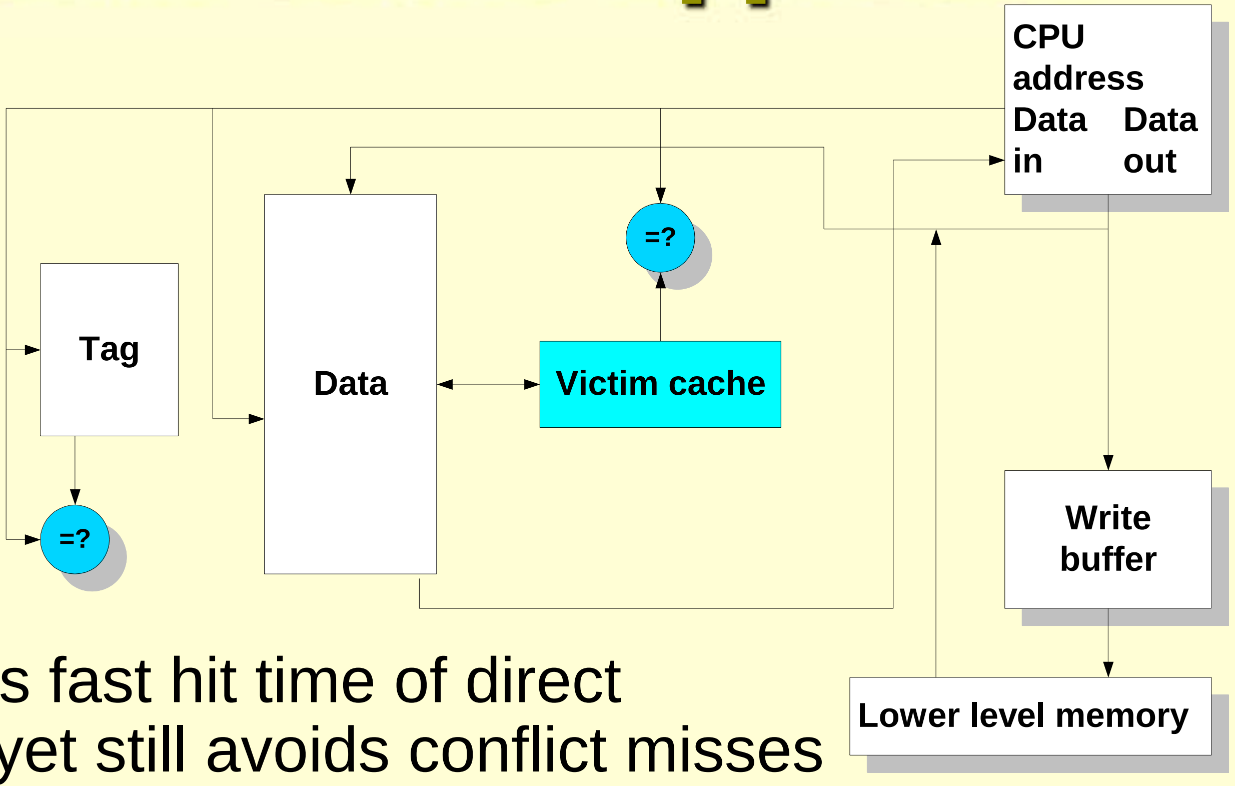
Assume hit time is 1 clock cycle and average miss penalty is 50 clock cycles for a direct mapped cache. The clock cycle increases by a factor of 1.10 for 2-way, 1.12 for 4-way, 1.14 for 8-way associative cache. Compare the average memory access based on the previous figure miss rates

Cache Size (KB)	Associativity			
	1-way	2-way	4-way	8-way
1	7.65	6.60	6.22	5.44
2	5.90	4.90	4.62	4.09
4	4.60	3.95	3.57	3.19
8	3.30	3.00	2.87	2.59
16	2.45	2.20	2.12	2.04
32	2.00	1.80	1.77	**1.79**
64	1.70	1.60	1.57	**1.59**
128	1.50	1.45	1.42	**1.44**

A good size of direct mapped cache can be very efficient given its simplicity

High associativity becomes a negative aspect

Victim Cache Approach



- Combines fast hit time of direct mapped yet still avoids conflict misses
  - Adds small fully asssociative cache between the direct mapped cache and memory to place data discarded from cache
  - Jouppi [1990]: 4-entry victim cache removed 20% to 95% of conflicts for a 4 KB direct mapped data cache
  - Technique is used in Alpha, HP machines and does not impair the clock rate

Pseudo-Associativity Mechanism

- Combine fast hit time of Direct Mapped and lower conflict misses of 2-way set associative
- Divide cache: on a miss, check other half of cache to see if there, if so have a pseudo-hit
- Simplest implementation inverts the index field MSB to find the other pseudo set
- To limit the impact of hit time variability on performance, swap block contents
- Drawback: CPU pipeline is hard if hit takes 1 or 2 cycles
  - Better for caches not tied directly to processor (L2)
  - Used in MIPS R1000 L2 cache, similar in UltraSPARC

H/W Pre-fetching of Instructions & Data

- Hardware pre-fetches instructions and data while handing other cache misses
  - Assume pre-fetched items will be referenced shortly
- Pre-fetching relies on having extra memory bandwidth that can be used without penalty

$$\text{Average memory access time} = \text{Hit time} + \text{Miss Rate} \times (\text{Prefetch hit rate} + (1 - \text{Prefetch hit rate}) \times \text{Miss penalty})$$

- Examples of Instruction Pre-fetching:
  - Alpha 21064 fetches 2 blocks on a miss
  - Extra block placed in “stream buffer”
  - On miss check stream buffer
- Works with data blocks too:
  - Jouppi [1990] 1 data stream buffer got 25% misses from 4KB cache; 4 streams got 43%
  - Palacharla & Kessler [1994] for scientific programs for 8 streams got 50% to 70% of misses from 2 64KB, 4-way set associative caches

Software Pre-fetching Data

- Uses special instructions to pre-fetch data:
  - Load data into register (HP PA-RISC loads)
  - Cache Pre-fetch: load into cache (MIPS IV, PowerPC, SPARC v. 9)
- Special pre-fetching instructions cannot cause faults (undesired exceptions) since it is a form of speculative execution
- Makes sense if the processor can proceeds without blocking for a cache access (lock-free cache)
- Loops are typical target for pre-fetching after unrolling (miss penalty is small) or after applying software pipelining (miss penalty is large)
- Issuing Pre-fetch Instructions takes time
  - Is cost of pre-fetch issues < savings in reduced misses?
  - Higher superscalar reduces difficulty of issue bandwidth

```
for (i = 0; i < 3; i = i+1)
 for (j = 0; j < 100; j = j+1)
  a[i][j] = b[j][0] * b[j+1][0];
```

$\rightarrow$

```
for (j = 0; j < 100; j = j+1)
    pre-fetch (b[i+7][0]);
    a[0][j] = b[j][0] * b[j+1][0];
    for (i = 1; i < 3; i = i+1)
       pre-fetch (a[i][j+7]);
       a[i-1][j] = b[j][0] * b[j+1][0];
```