Lecture 13: September 6, 2021

Computer Architecture and Organization-II

Biplab K Sikdar

Reducing miss rate

0.3.1 Block size

1. **Block size is very small (Figure 14), say a word forms a block**

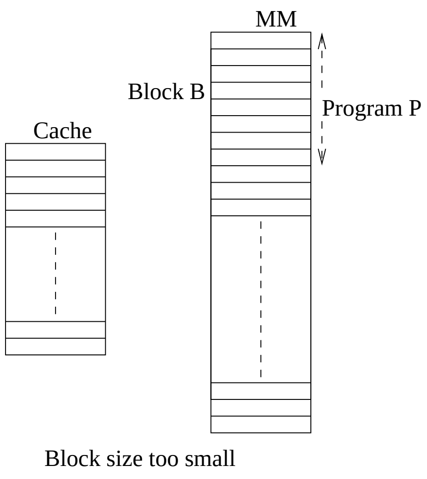

Figure 14: Block size is very small

Then blocks per program is very high. Compulsory misses are also high.

In direct map, conflict misses are increased as number of blocks is increased.

2. **Block size is very large (Figure 15), say larger than a program size**

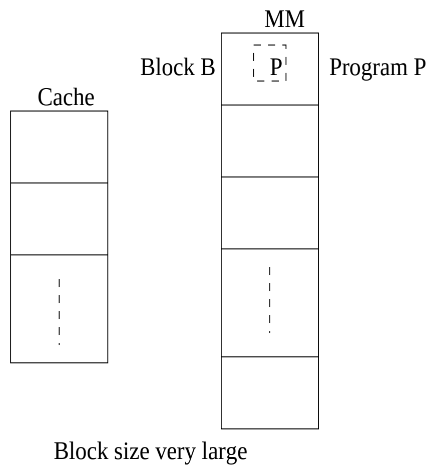

Figure 15: Block size is very large

Compulsory miss is effectively reduced to one.

Conflict misses increase as more blocks may map to same cache frame (considering existence of more than one running program in memory).

Effect of block size on miss rate is shown in Figure 16.

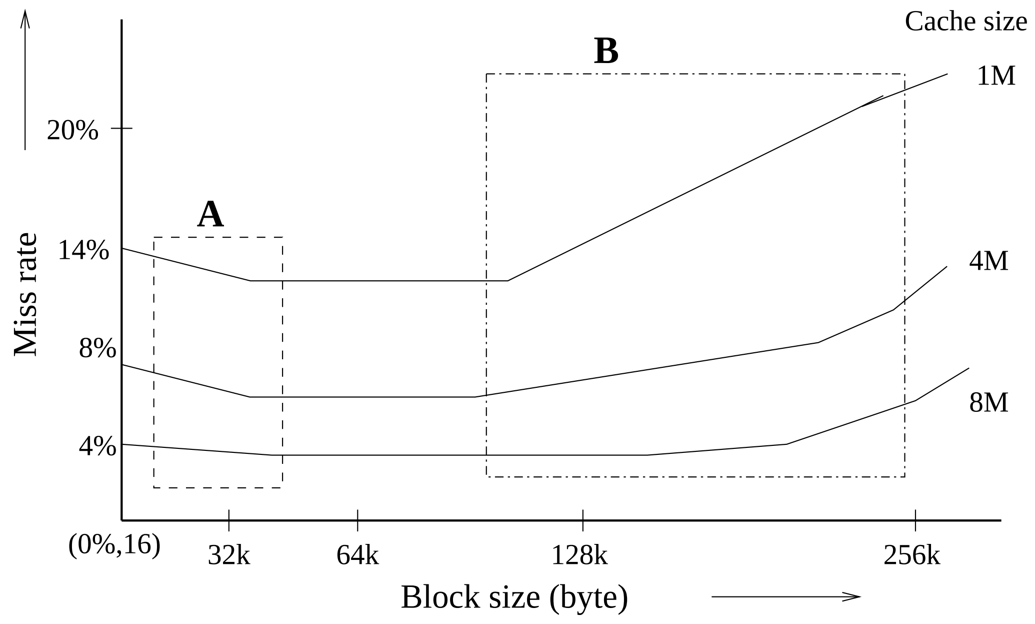

Figure 16: Block size vs miss rate

Simplest way of miss rate reduction is choice of large block size (**A** in the figure).

Implicit assumption is - fetch strategy is demand fetching)[1].

If block size is very large, then conflict miss dominates compulsory misses.

Further, capacity misses also increases (**B** in Figure 16).

(As number of blocks, of a program in execution, can be placed in cache is less.

[1]Fetch strategy - 1. Demand fetching and 2. Anticipatory fetching.

0.3.2 Cache size

One of the best options to reduce miss rate is through increase in cache size.

Keeping block size constant if cache size is increased,

1. Capacity misses are decreased,
2. Compulsory misses remain constant.

As number of compulsory misses is very less in comparison to other type of misses,

Capacity miss decreases ⇒ Significant reduction in miss rate (Figure 17).

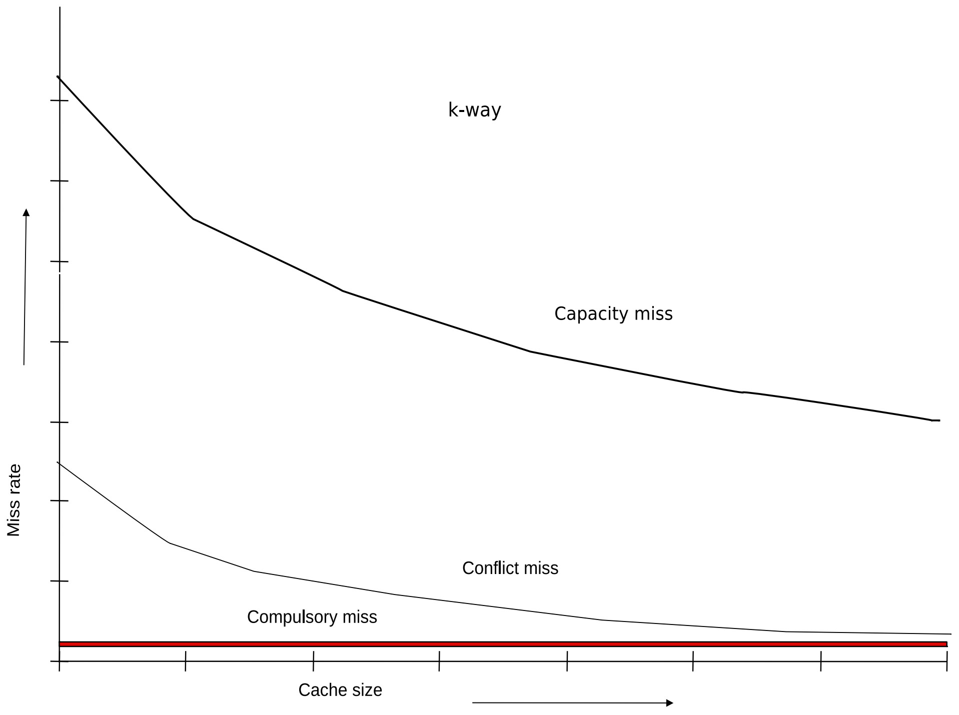

Figure 17: Cache size vs miss rate

Change in conflict miss rate is marginal while cache size increases.

Conflict misses increase for a k-way cache when k is decreasing.

0.3.3 Higher associativity

Higher associativity reduces conflict misses.

If it is k-way set-associative mapping, then miss rate improves for large k.

It is observed that

1. 8-way set-associative cache has miss rate as that of fully associative cache.
2. Direct mapped cache of m blocks has almost same miss rate as that of 2-way set-associative cache of $\frac{m}{2}$ blocks.

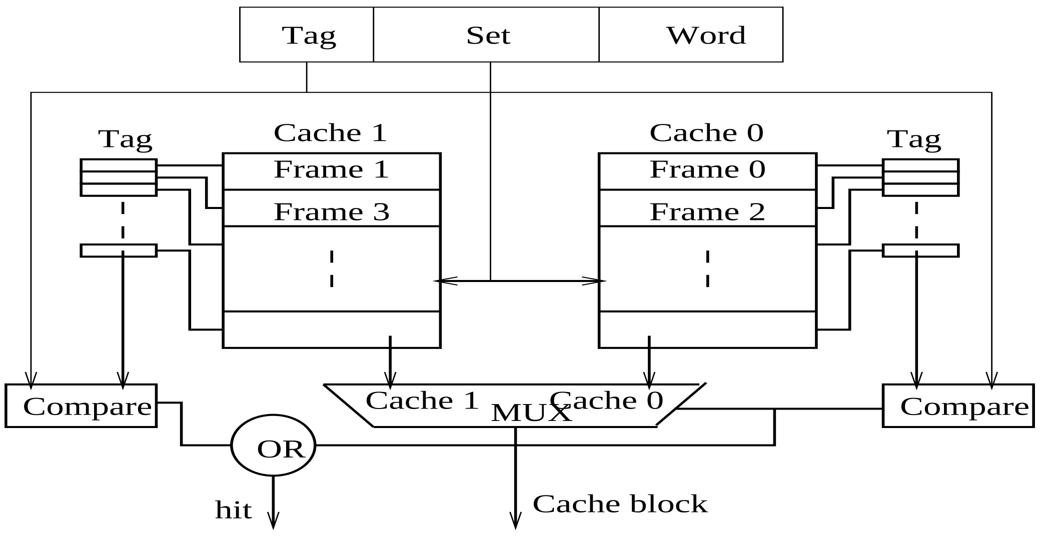

Figure 18: Set associative mapping

An efficient design is shown in Figure 18.

0.3.4 Victim caches

Is a small associative cache - placed in between cache and its refill path (Figure 19).

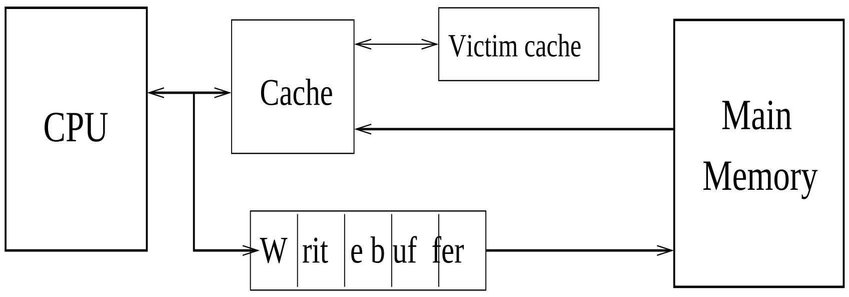

Figure 19: Victim cache

It contains only blocks that are discarded from a cache because of a miss (victim).

For next miss, system traces victim cache to check whether it contains desired block.

If found, victim block is swapped with a cache block.

If not, the block reference is a miss.

Victim cache doesn't affect clock cycle time/ miss penalty. It reduces conflict miss.

0.3.5 Pseudo-associative cache

This reduces conflict miss (Figure 20).

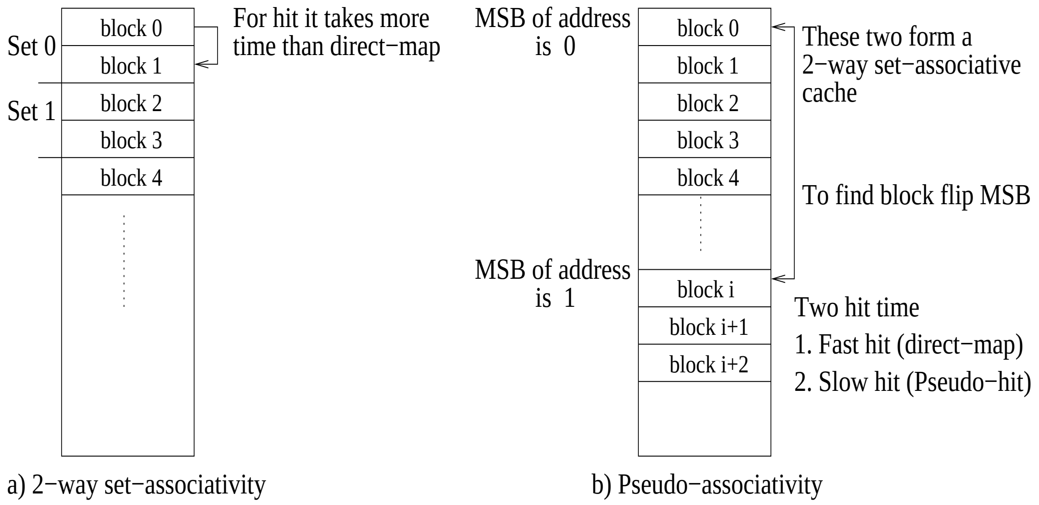

Figure 20: Pseudo-associative cache

It is introduced to ensure miss rate as that of 2-way associative caches.

Hit time in such a system is almost same as that of a direct-mapped cache.

Address of first cache frame is received from direct mapping scheme.
Second address (if required) is computed by complementing msb of the address.

In pseudo-associative cache, there are two types of hit.

1. Direct hit,
2. Slow hit (pseudo-hit).

0.3.6 Hardware prefetching

Prefetching of data/instructions is done in cache or in external buffer (stream buffer).

Normally, CPU access from buffer is faster than that from MM.

Access to such buffer is considered as hit - that effectively reduces miss rate.

Example: On cache miss for block B_1, fetch B_1 and next consecutive block (B_2).

That is block B_2 is prefetched and placed in stream buffer.

In a subsequent request if CPU demands B_2,

1. Cache request for B_2 is cancelled,
2. B_2 is read from stream buffer and
3. The next prefetch request is issued.

In a system (Figure 21), there can be

1. Instruction stream buffer beyond I-cache and
2. Multiple data stream buffers beyond D-cache.

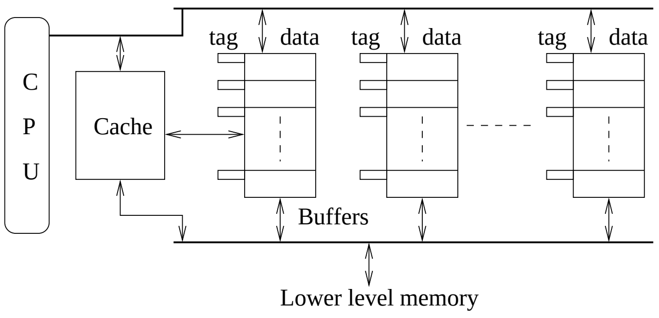

Figure 21: Stream buffers

Hardware prefetching requires high memory bandwidth

to handle simultaneous transfer of more than one block from MM.

A stream buffer is a LIFO.

If block b (referred) is not in cache but is in first entries of a stream buffer,

a new stream buffer is allocated and b's successors are prefetched in that.