CS223 : Computer Architecture & Organization

Lecture 22 [31.03.2022]

Advanced Cache Optimizations

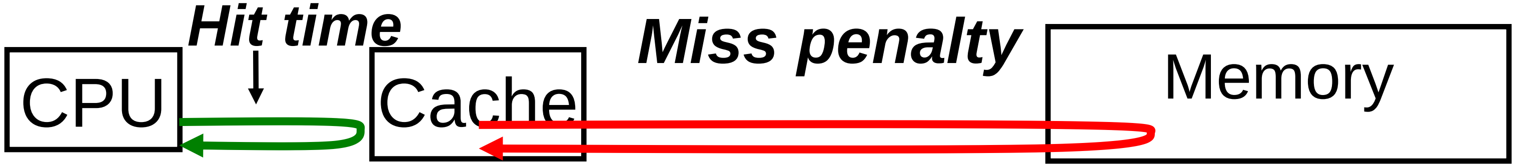
Dr. John Jose

Associate Professor

Department of Computer Science & Engineering

Indian Institute of Technology Guwahati, Assam.

Accessing Cache Memory



Average memory access time = Hit time + (Miss rate × Miss penalty)

- **Hit Time:** Time to find the block in the cache and return it to processor *[indexing, tag comparison, transfer].*
- **Miss Rate:** Fraction of cache access that result in a miss.
- **Miss Penalty:** Number of additional cycles required upon encountering a miss to fetch a block from the next level of memory hierarchy.

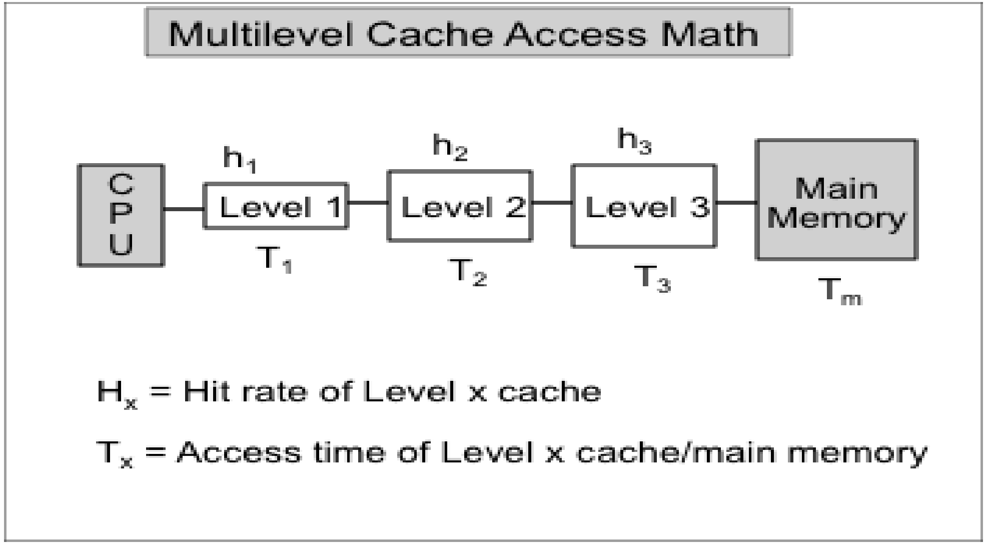
How to optimize cache ?

- Reduce Average Memory Access Time
- AMAT= Hit Time + Miss Rate x Miss Penalty
- Motives
    - Reducing the miss rate
    - Reducing the miss penalty
    - Reducing the hit time

Multilevel caches

- **Multilevel caches to reduce miss penalty**
- **Caches should be faster** to keep pace with the speed of processors, **AND** **cache should be larger** to overcome the widening gap between the processor and main memory
- Add another level of cache between the cache and memory.
- The first-level cache (L1) can be small enough to match the clock cycle time of the fast processor. [Low hit time]
- The second-level cache (L2) can be large enough to capture many accesses that would go to main memory, thereby lessening the effective miss penalty. [Low miss rate]

Multilevel caches



Multilevel Cache Access Math

H_x = Hit rate of Level x cache

T_x = Access time of Level x cache/main memory

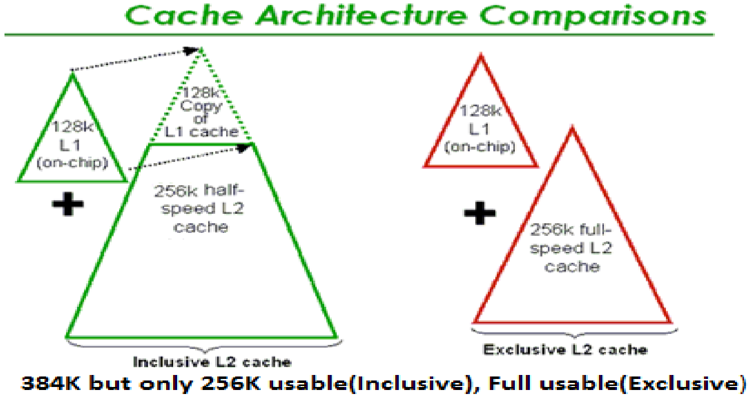
$$\text{Average memory access time} = \text{Hit time}_{L1} + \text{Miss rate}_{L1} \times \text{Miss penalty}_{L1}$$

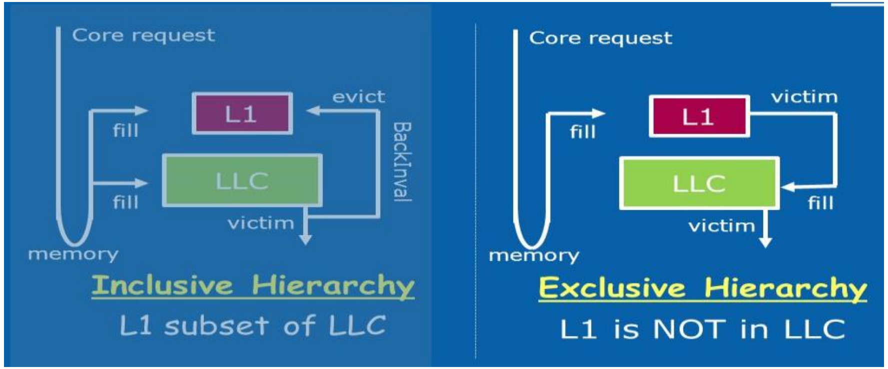
$$\text{Miss penalty}_{L1} = \text{Hit time}_{L2} + \text{Miss rate}_{L2} \times \text{Miss penalty}_{L2}$$

$$\begin{aligned}\text{Average memory access time} = {} & \text{Hit time}_{L1} + \text{Miss rate}_{L1} \\ & \times (\text{Hit time}_{L2} + \text{Miss rate}_{L2} \times \text{Miss penalty}_{L2})\end{aligned}$$

Multilevel caches

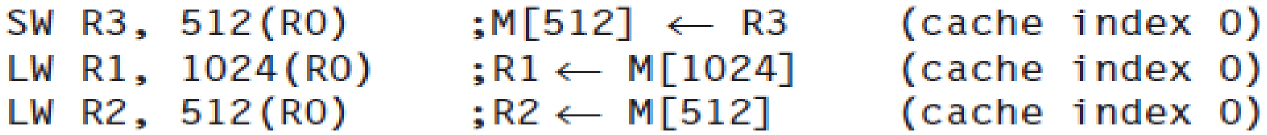
- **Multilevel caches to reduce miss penalty**
- **Local miss rate:** Number of misses in a cache level divided by number of memory access to this level.
- **Global miss rate:** Number of misses in a cache level divided by number of memory access generated by the CPU.
- **Inclusive and Exclusive caches**





Prioritize read miss over writes

- **Prioritize read misses to reduce miss penalty**
- If a read miss has to evict a dirty memory block, the normal sequence is write the dirty block to memory and read the missed block



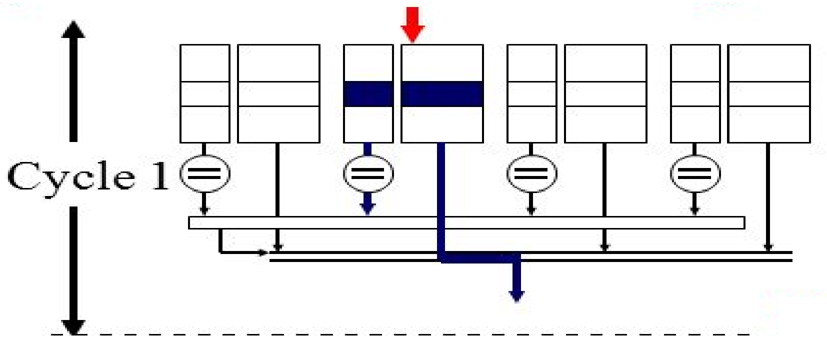
```
SW R3, 512(R0)      ;M[512] ← R3      (cache index 0)
LW R1, 1024(R0)     ;R1 ← M[1024]     (cache index 0)
LW R2, 512(R0)      ;R2 ← M[512]      (cache index 0)
```

- Optimization: copy the dirty block to a buffer, read from memory and then write the block - reduces CPU’s waiting time on read miss

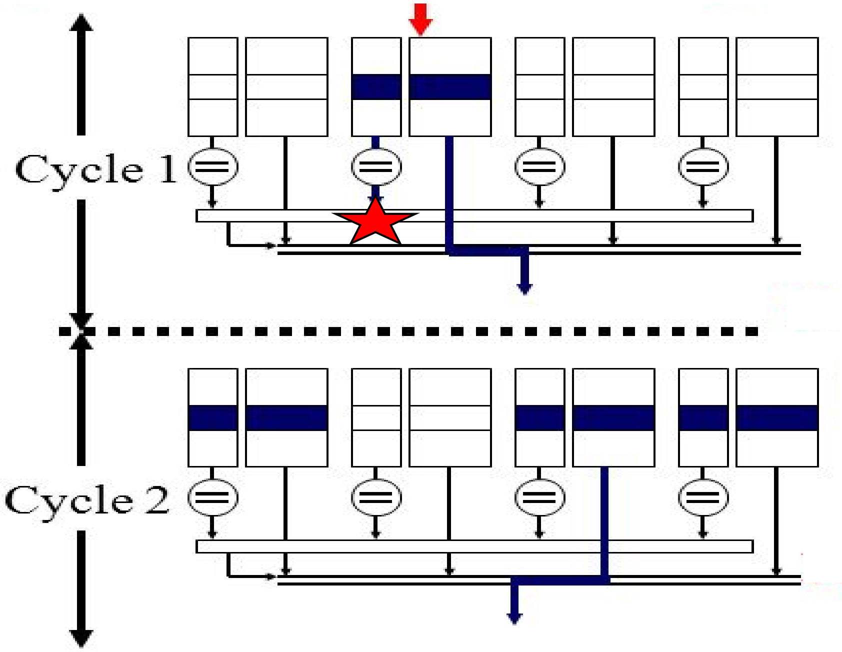
Way Prediction

- **Predict the way in a set to reduce hit time**
- To improve hit time, predict the way to pre-set multiplexer
- Extra bits are set to predict the block with in the set
- Remember the MRU way of a given set
- Using the prediction bits, power gating can be done on unused ways for reducing power.

Way Prediction



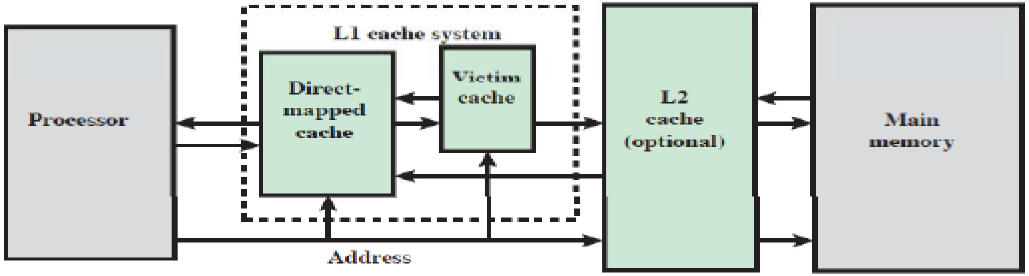
Way Prediction - Correct



Way Prediction Miss

Victim Caches

- **Introduce victim caches to reduce mis- penalty**
- Additional storage near L1 for MR evicted blocks
- Efficient for thrashing problem in L1 cache
- Look up Victim Caches before L2
- L1 and Victim caches are exclusive

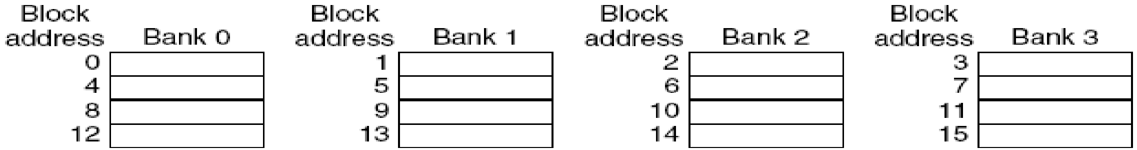


Pipelining Cache

- **Pipelined cache for faster clock cycle time.**
- Split cache memory access into several sub-stages
- Ex: Indexing, Tag Read, Hit/Miss Check, Data Transfer
- Pipeline cache access to improve bandwidth
  - Examples: Pentium 4 – Core i7: 4 cycles
- But slow in hits.
- Increases branch mis-prediction penalty
- Makes it easier to increase associativity

Multi-banked Caches

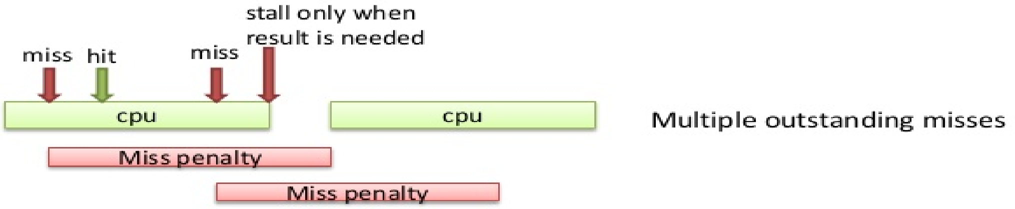
- **Multi-banked caches to increase cache bandwidth**
- Rather having a single monolithic block, divide cache into independent banks that can support simultaneous accesses.
  - ARM Cortex-A8 supports 1-4 banks for L2
  - Intel i7 supports 4 banks for L1 and 8 banks for L2
- Sequential Interleaving



Four-way interleaved cache banks using block addressing. Assuming 64 bytes per blocks, each of these addresses would be multiplied by 64 to get byte addressing.

Non-blocking Caches

- **Non blocking caches to increase cache bandwidth**
- Caches can serve hits under multiple cache miss in progress
  - *(a) Hit under miss (b) Hit under multiple miss*



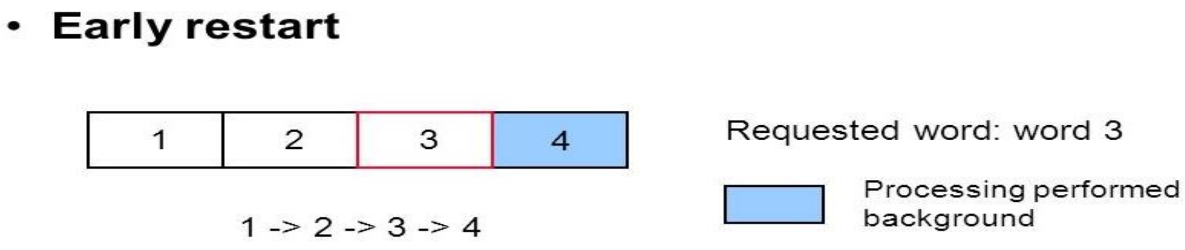
- Must needed for OOO superscalar processor for IPC increase
- L2 must support it with L1-MSHR (Miss Status Holding Reg.)
- On an L1 miss allocate MSHR entry, clear upon L2 respond with cache block reply.

Non-blocking Caches

- **Non blocking caches to increase cache bandwidth**
- Processors can hide L1 miss penalty but not L2 miss penalty
- Reduces the effective miss penalty by overlapping miss latencies
- Significantly increases the complexity of the cache controller as there can be multiple outstanding memory accesses
- Requires pipelined or banked memory system

Early Restart

- **Early restart to reduce miss penalty**
- Do not wait for entire block to be loaded for restarting CPU
- **Early restart**
  - Request words in normal order
  - Send missed work to the processor as soon as it arrives
  - L2 controller is not involved in this technique



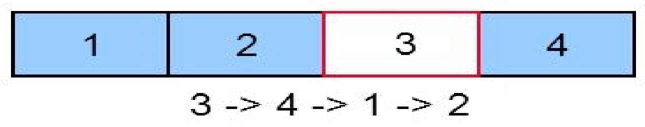
- **Early restart**

Requested word: word 3

Processing performed background

Critical Word First

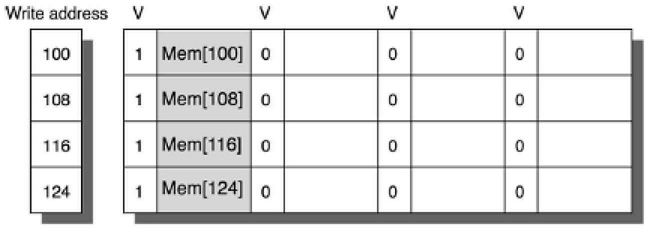
- **Critical word first to reduce miss penalty**
- **Critical word first**
  - Request missed word from memory first
  - Send it to the processor as soon as it arrives
  - Processor resume while rest of the block is filled in cache
  - L2 cache controller forward words of a block out of order.
  - L1 cache controller should re-arrange words in block
    - **Critical word first**



As soon as 3 is brought it is forwarded to CPU
In background 4, 1 and 2 is brought

Merging Write Buffer

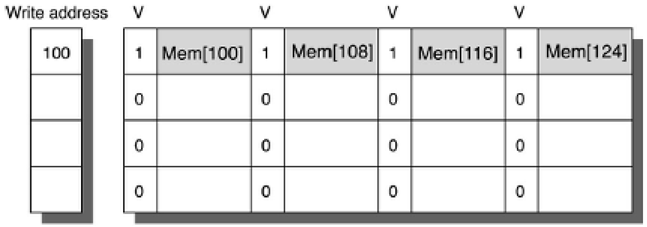
- **Write buffer merging to reduce miss penalty**
- Write buffer allows the processor to continue without waiting for writes to get over.
- When performing a store/write on a block that is already pending in the write buffer, update write buffer
- Reduces stalls due to full write buffer, improve buffer efficiency
- If buffer is full writes incur processor stall.



Write address	V		V		V		V	
100	1	Mem[100]	0		0		0	
108	1	Mem[108]	0		0		0	
116	1	Mem[116]	0		0		0	
124	1	Mem[124]	0		0		0	

•Each 64-bit word takes up one entry

No write buffering



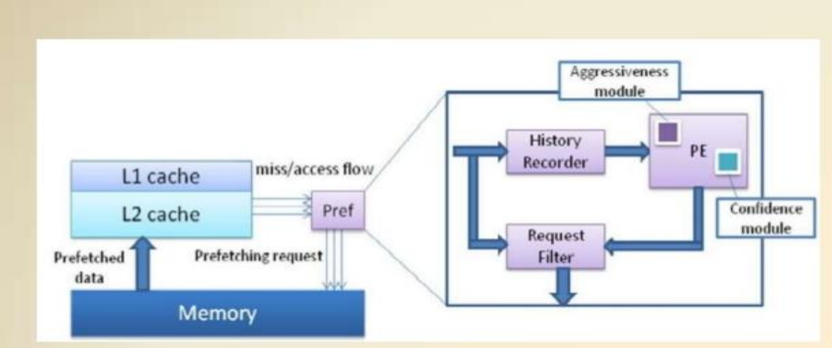
Write address	V		V		V		V	
100	1	Mem[100]	1	Mem[108]	1	Mem[116]	1	Mem[124]
	0		0		0		0	
	0		0		0		0	
	0		0		0		0	

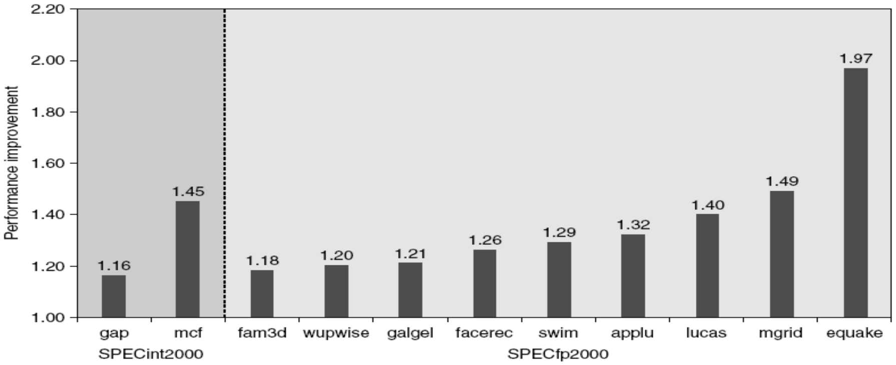
•Sequential addresses can be placed in one entry

Write buffering

Hardware Prefetching

- **Pre-fetching to reduce miss rate and miss penalty.**
- Pre-fetch items before processor request them.
- Fetch more blocks on miss -include next sequential block
- Requested block is kept in I-cache and next in stream buffer.
- If a missed block is in stream buffer, cache miss is cancelled





Compiler Controlled Pre-fetching

- **Pre-fetching to reduce miss rate and miss penalty.**
- Insert pre-fetch instructions before data is needed
- Pre-fetching will give performance only if processor reads from caches and executes while pre-fetching is in progress.
- Register pre-fetch
    - Loads data into register
- Cache pre-fetch
    - Loads data into cache
- Use loop unrolling and scheduling for pre-fetching data of adjacent iterations

Reference

- **Computer Architecture-A Quantitative Approach** (5th edition),

  John L. Hennessy, David A. Patterson, Morgan Kaufman.

- Chapter 2: Memory Hierarchy Design
  - Section 2.1: Introduction
  - Section 2.2: Ten Advanced Cache Optimizations of Cache Performance
- **NPTEL Video Links:**
  - **https://tinyurl.com/yf94dnby**
  - **https://tinyurl.com/yfgmfsmm**

johnjose@iitg.ac.in

http://www.iitg.ac.in/johnjose/