Improving Cache Performance

**AMAT**: Average Memory Access Time

\[ \text{AMAT} = T_{hit} + \text{Miss Rate} \times \text{Miss Penalty} \]

- **Small Hit Time**: On the critical (common case) path
  - Requires *small, direct-mapped* cache
    - Small *size* and lack of *associativity* implies higher miss rate

- **Compensate by reducing Miss Penalty**
  - **Structural**: Multi-level caches, Critical word/Early Restart,
  - **Latency Hiding**: Using *concurrency* to reduce miss rate or miss penalty
Techniques for Reducing Miss Penalty

Effect of Miss Penalty in clock cycles increases with faster processors

Two-level caching

- **Second-level (L2) cache** between the primary cache and memory

  - **Primary (L1) cache**: small and matches processor cycle time (miss rate may be higher)
  - **Miss Penalty is small** since misses fielded by L2 cache (rather than main memory)

- Secondary (L2) cache large enough to reduce miss ratio to memory

\[ m_2 m_1 N \]: Global Miss Rate of L2

\[ m_2 \]: Local Miss Rate of L2

\[ m_1 \]: Miss Rate of L1
**L2 cache**

Local Miss Rate: Fraction of requests made to a cache that miss

Global miss rate: Fraction of requests made by the processor that miss

L1: Local miss rate = Global Miss rate = \( m_1 \)
L2: Local miss rate = \( m_2 \)

Fraction of requests made to L2 = Local miss rate of L1 = \( m_1 \)

Global miss rate = \( m_2 \times m_1 \)

**AMAT** = Hit time (L1) + Miss Rate (L1) x Miss Penalty (L1)

Miss Penalty (L1) = Hit time (L2) + Local Miss Rate (L2) x Miss Penalty (L2)

Local Miss Rate (L2): \( m_2 \)

Fraction of references to the L2 cache that are not found in L2

Relatively high (cream skimmed by L1 cache)

**AMAT** = Hit time (L1) + \( m_1 \times [ \text{Hit time (L2)} + m_2 \times \text{Miss Penalty (L2)} ] \)

**AMAT** = Hit time (L1) + \( m_1 \times \text{Hit Time (L2)} + m_1 \times m_2 \times \text{Miss Penalty (L2)} \)

Stalls/Instruction = Misses/Instruction x Hit Time (L2) + Global Misses/Instruction (L2) x Miss Penalty (L2)
L2 cache

Speed of L1 cache affects cycle time (lean and mean)
Speed of L2 cache affects Miss Penalty of L1 cache

Reduce Miss Rate of L2 cache

Large: Reduce Miss rate due to capacity/conflict

Higher Associativity

• Inclusion Principle
  • L1 data are always present in L2
  • Not in L2 implies Not in L1
  • Cache coherence: Multiprocessor (or I/O processor) snooping L2 cache does not have to search L1 if block not present in L2

• Block Size Mismatch
  • L2 block size > L1 block size
  • To maintain inclusion several L1 blocks may need to be invalidated if an L2 block is invalidated (or replaced)
  • May increase Miss rate of L1 cache
L2 cache

- **Inclusion Principle**
  - L1 data are always present in L2
  - Not in L2 implies Not in L1
  - **Cache coherence**: Multiprocessor (or I/O processor) snooping L2 cache does not have to search L1 if block not present in L2
Techniques for Reducing Miss Penalty

2. Early Restart and Critical Word First

- Processor resumes execution as soon as desired word from block available
- Access memory so that missed (critical) word is accessed and transferred first
- Most beneficial when:
  - Block size is large so miss penalty is high
  - Immediate access to non-critical words of the block not likely
Techniques for Reducing Miss Penalty

4. **Sub-blocking**: Treat a block as made up of several sub blocks

   - Large block size increases miss penalty (-)
     - **Block**: Unit associated with tag
     - On a **miss** only a sub block (containing the missed word) is read.
     - Remaining sub blocks of the block are marked as invalid
     - Tag match does not necessarily imply a sub block hit
     - Storage saved by having one bit (Valid/Invalid) per sub block instead of full tag
Techniques for Reducing Miss Penalty

5. Victim Cache: Small (specialized) Fully Associative cache

- Holds (only) blocks that have been replaced from the cache (victims)
- Check victim cache for missed block and swap with cache if found
  - Simulates a larger associativity without increasing size of main cache (shared by all sets incurring conflicts) and corresponding increase in cycle time for cache (hit) access
- Useful when main cache is small
Techniques for Reducing Miss Penalty

5. Giving Read Priority over Writes
   • Write Through cache policy
   • Write Buffer to hold pending writes
   • Give reads priority over pending writes
     • Problem: May cause RAW hazards to memory if read location in write buffer
       • Need to check write buffer for potential hazard
   • Write Back cache policy
     • An evicted dirty block is written to memory and new block read from memory
     • Write buffer: copy evicted block from cache to write buffer
     • Either stall or check for address match if write buffer not empty on read miss

6. Merging Write Buffer
   • Consolidate outstanding writes in the write buffer
   • Only the most recent write to an address
   • Arrange words in units of blocks; managing dirty sub-blocks
Techniques for Reducing Hit Time

1. Small Simple Caches

2. Pipelining Writes for Fast Write Hits
   - Tag check and data access cannot occur in parallel for writes
   - Pipeline the stages: Save in write buffer.
   - Update cache on next write or cache miss.
   - Reads must check the buffer for latest copy.

3. Avoiding Address Translation before Cache Indexing (virtual caches)

   Why not use virtual caches?
   (a) Cache needs to be flushed on context switch
       -- use PID as extension of the address tag

   (b) Aliasing: Multiple virtual addresses for same physical address
       Inconsistency between cached copies of the same physical location

       ---- Restrict aliased addresses in some way e.g. last n bits identical
           Direct Mapped cache of size $2^n$ will map these to same cache location

   (c) I/O: Typically uses physical addresses
       Would need translation to deal with virtual cache

Clearer after Virtual Memory Discussion
General Techniques

1. Prefetching Techniques

(a) **Hardware Prefetching**: 
Fetch block from memory before it is requested by the program 
Memory access overlapped with program execution 
Can use for instructions or data: instruction prefetch more predictable 
Prefetch directly into cache or external buffer

**Instruction Stream Buffer**: 
On I-cache miss the requested block and next consecutive block fetched 
Requested block placed in cache 
Prefetched block in Instruction Stream Buffer (ISB) 
If requested block found in ISB moved to cache and only prefetch is issued

**Example**: Assume hit time of 2 cycles and I-cache miss rate of 1.1%. Prefetch hit rate is assumed to be 25%, the miss penalty to memory is 50 clock cycles and the miss penalty to ISB is 1 cycle.

\[ T_{avg} = 2 + (1.1\% \times 25\% \times 1) + (1.1\% \times 75\% \times 50) = 2.415 \]

Effective miss rate: \( (2.415 - 2)/50 = 0.83\% \)
Decreasing Miss Rate / Miss Penalty

2. **Compiler-Controlled Prefetching:**
   - Explicit instructions to prefetch block from memory
   - Compiler inserts prefetch instructions based on program analysis
   - Cache prefetch or register prefetch (destination of load is cache/register)
   - Non-faulting prefetch: Ignored if it will cause exceptions (nonbinding prefetch)
   - Requires non-blocking (lockup free) cache:
     - continue providing cached data while stalled for prefetch

3. **Nonblocking Caches:** *Lockup-free cache*

   Processors exploiting ILP can benefit from out-of-order data accesses
   - Hit-under miss: Permit cache access while servicing a miss (or multiple misses)
   - Cache controller gets complex