Caches and Prefetching

Impact of Cache Misses
- Cache misses are expensive
  - Blocking cache: severely reduce performance
  - Non-blocking cache: instruction stalls in ROB, can prevent instruction issue
  - See paper Table 1-1
- Tradeoff between associativity and hit time
  - Direct-mapped cache: fast access time, more conflict misses
  - Set-associative cache: slower access, fewer misses
- In Jouppi’s paper, techniques were proposed to reduce misses in a direct-mapped cache
  - Baseline miss rate: Table 2-2
  - Performance: Figure 2-2

Miss Caching
- In direct-mapped caches, conflict misses represent significant percentage
  - Average: 39% for D-cache, 29% for I-cache (Figure 3-1)
- Miss Cache: Small cache placed between the L1 and L2 caches
  - Provides additional associativity without increasing hit time in common case
  - Fully associative cache containing 2-5 lines
  - On a miss, data is returned to both L1 cache and miss cache
  - Organization in Figure 3-2
- Results: Figure 3-3
  - More effective for D-cache than I-cache (discuss)

Victim Caching
- Disadvantage of Miss Cache: data redundancy
  - Needs at least two lines to be effective
- Victim Cache: On a miss, replacement victim line is placed in the victim cache
  - Provides additional associativity without increasing hit time in common case
  - Even a single line can be effective
  - Always an improvement over miss caching
  - Organization in Figure 3-4

Victim Caching (Cont.)
- Results: Figure 3-5
  - More effective for D-cache than I-cache
- Impact of cache size on victim cache performance: Figure 3-6
- Impact of cache line size on victim cache performance: Figure 3-7

Prefetching
- Bringing lines to the cache before being requested
  - Can reduce compulsory and capacity misses
- Requests to next level of cache are either:
  - Demand misses: fill request due to cache miss
  - Prefetch: fill request in anticipation of data request
- Instruction and data access patterns are different (discuss)
Prefetching Terminology

- **Timeliness**: Measures whether the prefetch arrives early enough to avoid a miss.
- **Even if miss is not totally avoided, miss latency is reduced**.
- **Prefetch Hit**: Prefetched line that was hit in the cache before being replaced (miss avoided).
- **Prefetch Miss**: Prefetched line that was replaced before being accessed.
- **Prefetch rate**: Prefetches per instruction (or 1000 inst.).
- **Accuracy**: Percentage of prefetch hits to all prefetches.
- **Coverage**: Percentage of misses avoided due to prefetching.
  - $100 \times \frac{\text{Prefetch Hits}}{\text{Prefetch Hits} + \text{Cache Misses}}$

Classification of Prefetched Lines

- **Useful Prefetch**
  - Prefetch hit before being replaced.
  - Results in avoiding a cache miss.
- **Useless Prefetch**
  - Prefetch is replaced before being accessed (prefetch miss).
  - Downside: Increases demand for cache bandwidth.
- **Harmful Prefetch**
  - Prefetch is replaced before being accessed AND prefetch replaces a line that is requested later (cache pollution).
  - Results in an additional cache miss.

Simple Prefetching Alternatives

- **Prefetch always**
  - Prefetch after every reference.
  - Leads to significant demand on resources for next level in cache hierarchy.
- **Prefetch on miss** (Also called one block lookahead)
  - On a miss, we prefetch the next sequential line as well.
  - Cuts #misses in a sequential stream in half.
  - We can also implement N-block lookahead.
- **Tagged Prefetch**
  - Each block has a tag status bit associated to it.
  - On a prefetch, tag bit set to zero.
  - On a hit, tag bit set to 1 (indicating prefetch hit).
  - When a block’s status bit changes from 0 to 1, next block is prefetched.

Stream Buffers

- **Tagged prefetch may not be timely if cache lines are consumed faster than they are prefetched**.
- **Need to start prefetching before a tag status bit transition takes place**.
- **Stream buffer organization**: Figure 4-2.

Stream Buffer Operation

- **On a cache miss**
  - Stream buffer prefetches successive lines starting at the miss address.
  - As each prefetch is sent out, we allocate an entry in the stream buffer and set available bit to false.
  - When prefetch data returns, it is placed in buffer entry; available bit set to true.
  - Prefetch lines are stored in the stream buffer not the cache to avoid cache pollution.
- **On a cache miss and buffer hit**
  - Data loaded from stream buffer in one cycle.
  - All buffer entries shift by one, new line prefetched to vacant entry.
- **On a non-sequential miss**
  - Stream buffer flushed.
  - Prefetching starts from new miss address (even if miss is present in another stream buffer entry).

Stream Buffer Performance

- **Performance**: Figure 4-3.
- **More successful for instructions compared to data**.
- **Multi-way stream buffer addresses data streams**
  - Organization: Figure 4-4.
  - Performance: Figure 4-5.
- **Sensitivity to cache size**: Figure 4-6.
- **Sensitivity to cache line size**: Figure 4-7.
**Stride-Based Prefetching (Chen & Baer)**

- Detect prefetching patterns based on load/store instruction PC
  - Strided access: A constant value is added to current address to compute next address
  - Example: A, A+20, A+40, A+60, ...
- Lookahead prefetching
  - Used to improve timeliness of prefetches
  - Uses a “lookahead PC”

**Reading Assignment**

- Midterm exam on Monday
  - Open notes, book, calculator
  - No laptops, PDAs or cell phones 😊
- Tuesday’s readings:
  - David B. Papworth, “Tuning the Pentium Pro Microarchitecture,” IEEE Micro 1996 (Skim)