CS510 Computer Architecture

Lecture 16:
Improving Cache Performance

Soontae Kim

Spring 2019

School of Computing, KAIST

Improving Cache Performance

1. Reduce the miss rate,

2. Reduce the miss penalty, **or**

3. Reduce the time to hit in the cache.

1. Reducing Miss Penalty: Read Miss Priority over Write

- **Write-through with write buffers offer RAW conflicts with main memory reads on cache misses**
  - If simply wait for write buffer to empty, might increase read miss penalty
  - or check write buffer contents before read; if no conflict, let the memory access continue
- **Write-back also want buffer to hold replaced blocks**
  - Read miss replacing dirty block
  - Normal: Write dirty block to memory, and then do the read
  - Instead copy the dirty block to a write buffer, then do the read, and then do the write
  - Less CPU stalls since restarts as soon as do read
  - Check the buffer before reading from memory or wait until the buffer is empty

2. Reduce Miss Penalty: Early Restart and Critical Word First

- **Don't wait for full block to be loaded before restarting CPU**
  - *Early restart*—As soon as the requested word of the block arrives, send it to the CPU and let the CPU continue execution
  - *Critical Word First*—Request the missed word first from memory and send it to the CPU as soon as it arrives; let the CPU continue execution while filling the rest of the words in the block. Also called *wrapped fetch* and *requested word first*
- **Generally useful in large blocks,**
- **Spatial locality may be a problem; tend to want next sequential word, so not clear if benefit by early restart**

block

3. Reduce Miss Penalty: Non-blocking Caches to reduce stalls on misses

- ***Non-blocking cache*** **or** ***lockup-free cache*** **allow data cache to continue to supply cache hits during a miss**
- **"*hit under miss*" reduces the effective miss penalty by working during miss vs. ignoring CPU requests**
- **"*hit under multiple miss*" or "*miss under miss*" may further lower the effective miss penalty by overlapping multiple misses**
  - Significantly increases the complexity of the cache controller as there can be multiple outstanding memory accesses
    - **Miss Status Handling Registers (MSHRs)**
  - Penium Pro allows 4 outstanding memory misses

Value of Hit Under Miss for SPEC

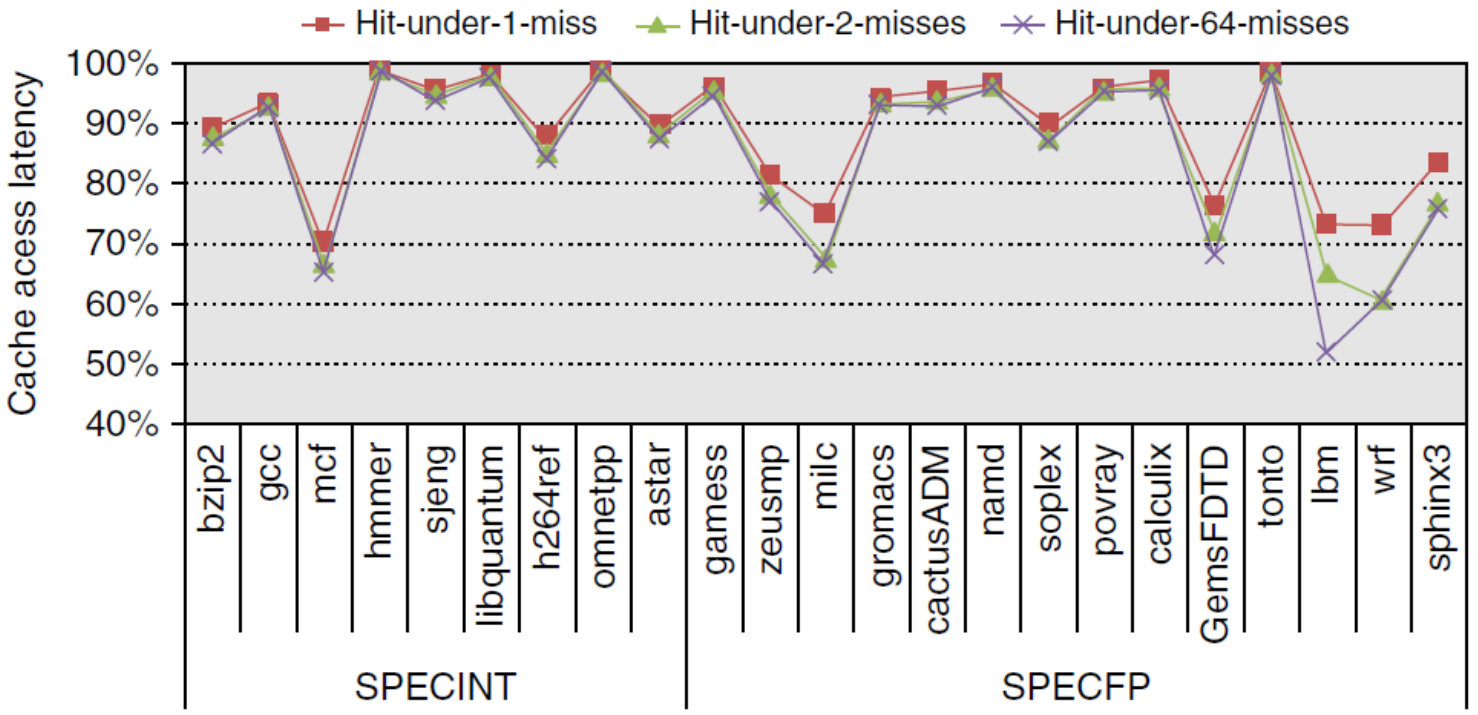
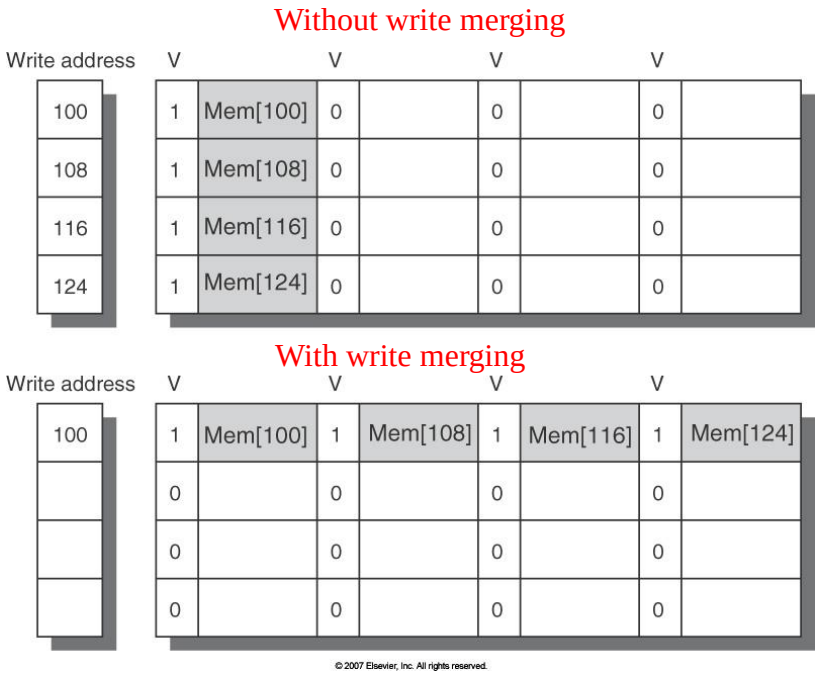


Figure 2.5 The effectiveness of a nonblocking cache is evaluated by allowing 1, 2, or 64 hits under a cache miss with 9 SPECINT (on the left) and 9 SPECFP (on the right) benchmarks. The data memory system modeled after the Intel i7 consists of a 32KB L1 cache with a four cycle access latency. The L2 cache (shared with instructions) is 256 KB with a 10 clock cycle access latency. The L3 is 2 MB and a 36-cycle access latency. All the caches are eight-way set associative and have a 64-byte block size. Allowing one hit

4. Merging write buffer

- Write-through caches require write buffer to reduce memory traffic
- Write-back caches also use write buffer to reduce miss penalty
- After data are written to WB, processor proceeds while WB takes care of writing to memory
- If new write data matches old data in WB, they are merged (***write merging***)
- If WB is full and there is no address match in WB, cache and processor are stalled until WB has an empty entry
  - ***Multiwords write to memory is faster than multiple single word writes***



Improving Cache Performance

1. Reduce the miss rate,

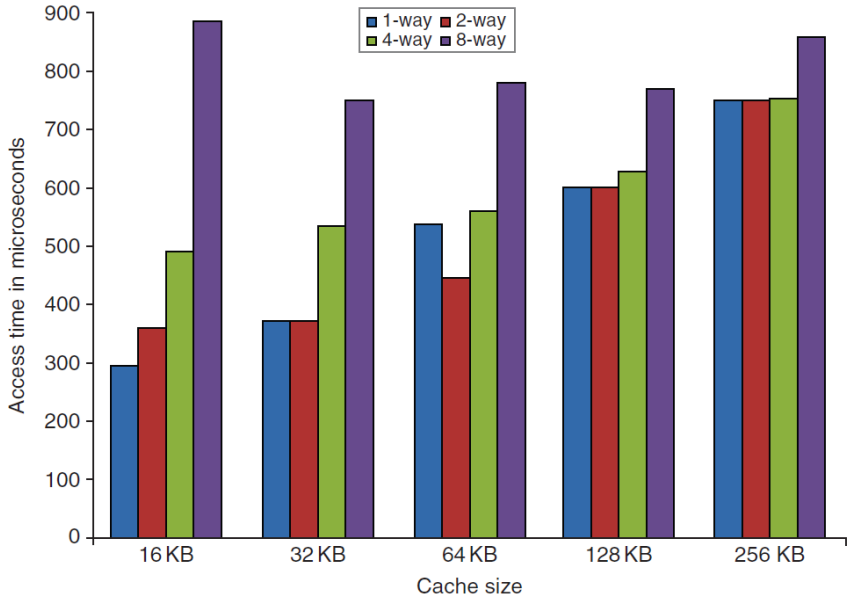
2. Reduce the miss penalty, or

***3. Reduce the time to hit in the cache*.**

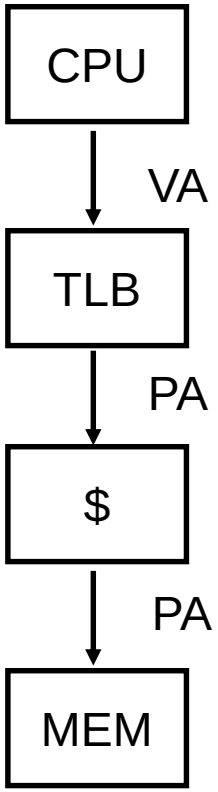
$$AMAT = HitTime + MissRate \times MissPenalty$$

1. Fast Hit times via Small and Simple Caches

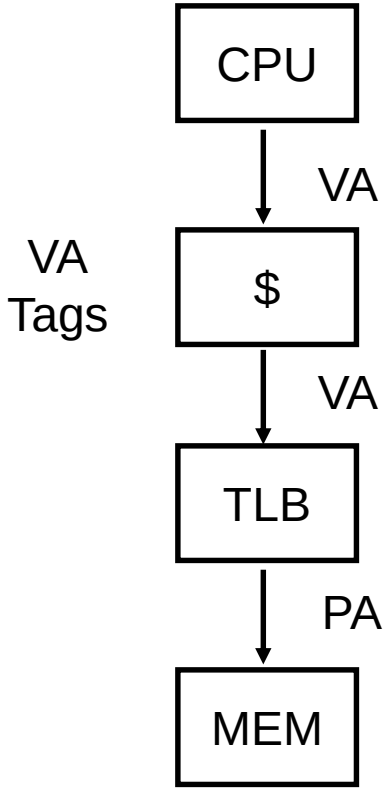
- **16KB in Pentium III to 8KB in Pentium 4**
  - Small data cache and fast clock rate
  - Use out-of-order execution and L2 cache
- **Direct mapped caches are faster than set-associative caches**



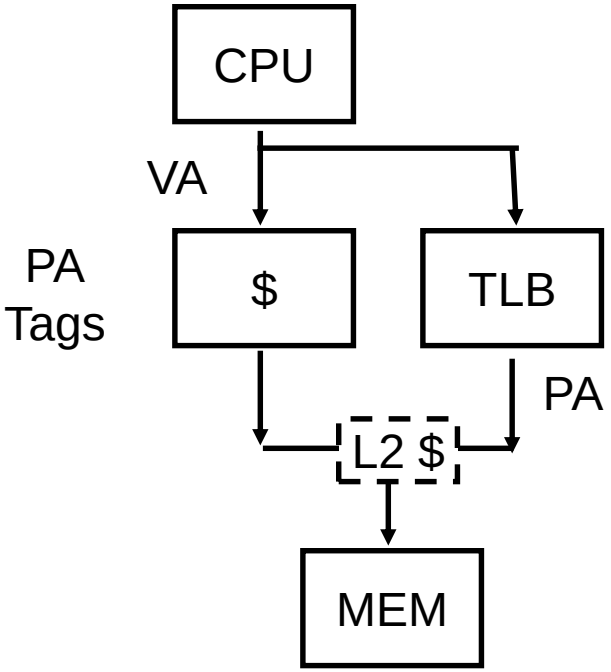
2. Fast hits by Avoiding Address Translation during indexing



Conventional Organization



Virtually Addressed Cache
Translate only on miss
Synonym Problem



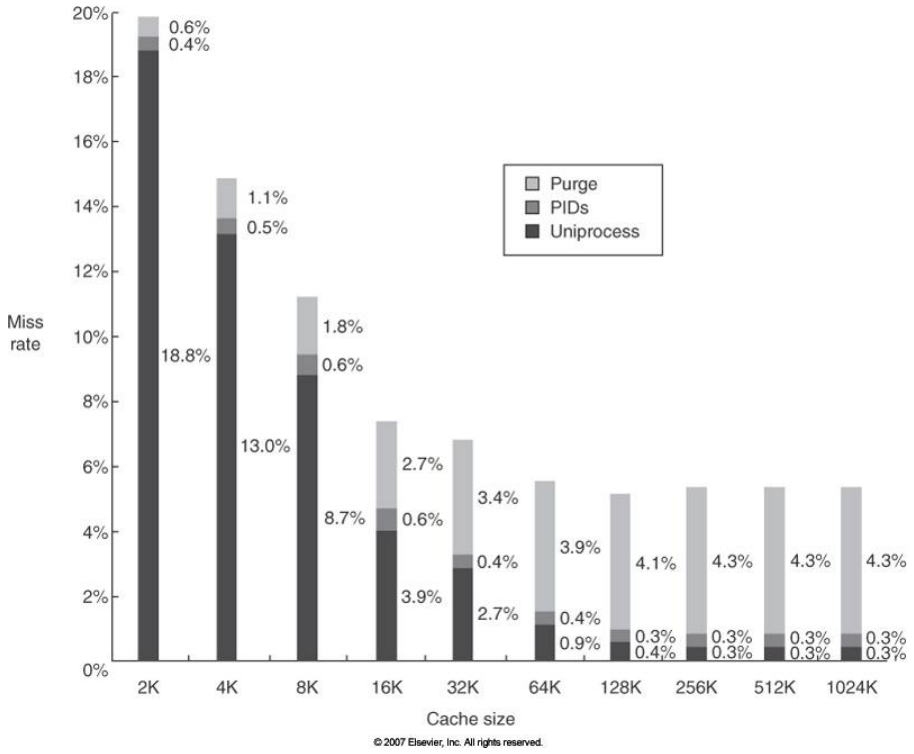
Overlap $ access with VA translation: requires $ index to remain invariant across translation

2. Fast hits by Avoiding Address Translation during indexing

- **Send virtual address to cache? Called *Virtually Addressed Cache* or just *Virtual Cache* vs. *Physical Cache***
  - Every time process is switched, the cache is flushed; otherwise get false hits
    - **Cost is time to flush + "compulsory" misses from empty cache**
  - Dealing with *aliases* (sometimes called *synonyms*); Two different virtual addresses map to same physical address; two copies of the same data in virtual cache
  - I/O must interact with cache, so need virtual address
- **Solution to cache flush**
  - Add *process identifier tag* that identifies process as well as address within process: can't get a hit if wrong process

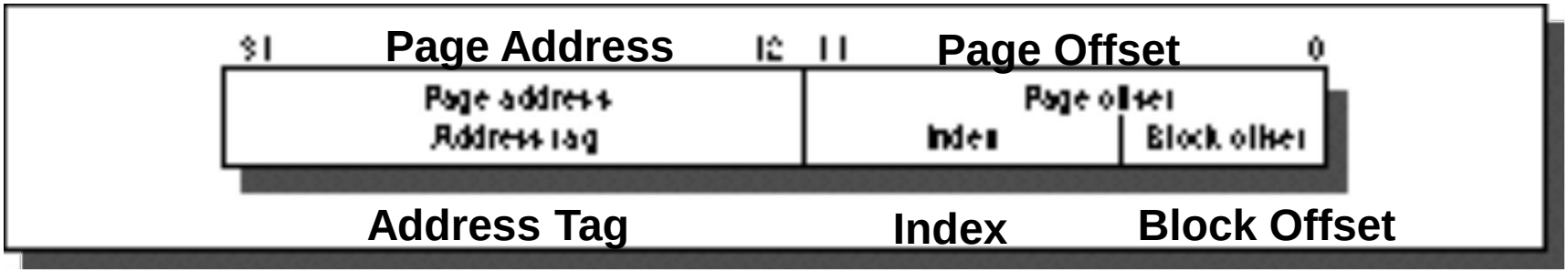
2. Fast Cache Hits by Avoiding Translation: Process ID impact

- **Black is uniprocess**
- **Dark Gray is multiprocess when use process ID tag**
- **Light Gray is multiprocess when flush cache**
- **Y axis: Miss Rates up to 20%**
- **X axis: Cache size from 2 KB to 1024 KB**



2. Fast Cache Hits by Avoiding Translation: Use page offset as index

- If index is part of physical address or does not change during translation, then can start tag access in parallel with translation, after which compare to physical tag
  - This is called virtually indexed physically tagged cache



- Limits cache to page size: what if want bigger caches and uses same trick?
  - Higher associativity
  - Page coloring

3. Pipelining & Multibanking Caches for high bandwidth

- **Pipelining**
  - Fast cycle time, high bandwidth and slow hits
  - Pentium 1 cycle, Pentium pro and III 2 cycles, Pentium4 4 cycles
  - Greater branch misprediction penalty and more load-use cycles
- **Multibanking**
  - AMD Opteron L2 cache has 2 banks
  - Sun T1 L2 cache has 4 banks