

Improving Cache Performance

1. Reduce the miss rate,
2. *Reduce the miss penalty*, or
3. Reduce the time to hit in the cache.

Reducing Miss Penalty: Read Priority over Write on Miss

- Write buffers may offer RAW conflicts with main memory reads on cache misses
- If simply wait for write buffer to empty might increase read miss penalty by 50%
- Check write buffer contents before read; if no conflicts, let the memory access continue
- Write Back?
 - Read miss may require write of 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
 - CPU stalls less since it can restart as soon as read completes

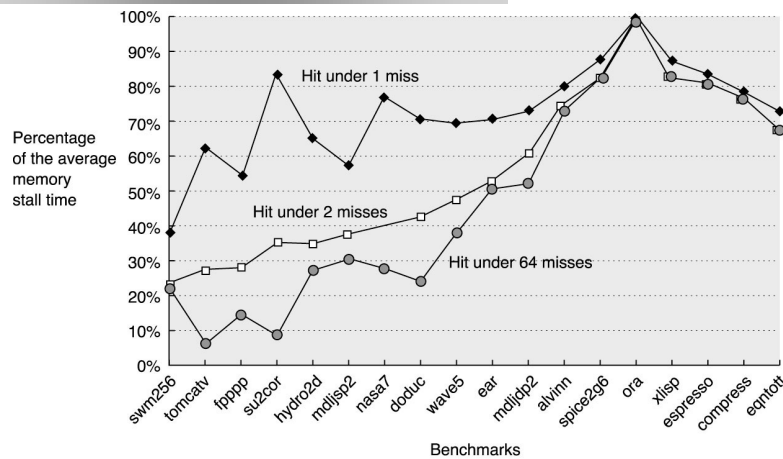
Early Restart and Critical Word First

- 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*
- Most useful with large blocks,
- Spatial locality a problem; often we next want the next sequential word soon, so not always a benefit (early restart).

Non-blocking Caches to reduce stalls on misses

- *Non-blocking cache* or *lockup-free cache* allowing the data cache to continue to supply cache hits during a miss
- “*hit under miss*” reduces the effective miss penalty by being helpful during a miss instead of ignoring the requests of the CPU
- “*hit under multiple miss*” or “*miss under miss*” can 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
- assumes “stall on use” not “stall on miss” which works naturally with dynamic scheduling, but can also work with static.

Value of Hit Under Miss for SPEC



CSE 240A

Dean Tullsen

Miss Penalty Reduction: Second Level Cache

- L2 Equations

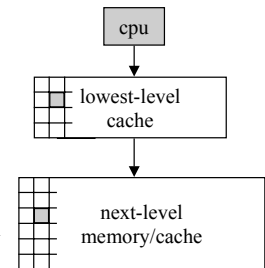
$$AMAT = Hit\ Time_{L1} + Miss\ Rate_{L1} \times Miss\ Penalty_{L1}$$

$$Miss\ Penalty_{L1} = Hit\ Time_{L2} + Miss\ Rate_{L2} \times Miss\ Penalty_{L2}$$

$$AMAT = Hit\ Time_{L1} + Miss\ Rate_{L1} \times (Hit\ Time_{L2} + Miss\ Rate_{L2} + Miss\ Penalty_{L2})$$

- Definitions:

- *Local miss rate*—misses in this cache divided by the total number of memory accesses *to this cache* ($Miss\ rate_{L2}$)
- *Global miss rate*—misses in this cache divided by the total number of memory accesses *generated by the CPU* ($Miss\ Rate_{L1} \times Miss\ Rate_{L2}$)



CSE 240A

Dean Tullsen

Multi-level Caches, cont.

- L1 cache local miss rate 10%, L2 local miss rate 40%. What are the global miss rates?
- L1 highest priority is fast hit time. L2 typically low miss rate.
- Design L1 and L2 caches in concert.
- Property of inclusion -- if it is in L1 cache, it is guaranteed to be in the L2 cache -- simplifies design of consistent caches.
- L2 cache can have different associativity (good idea?) or block size (good idea?) than L1 cache.
- Can be applied recursively to Multilevel Caches
 - Danger is that time to DRAM will grow with multiple levels in between

CSE 240A

Dean Tullsen

Reducing Miss Penalty Summary

- Four techniques
 - Read priority over write on miss
 - Early Restart and Critical Word First on miss
 - Non-blocking Caches (Hit Under Miss)
 - Multi-level Caches

CSE 240A

Dean Tullsen

Review: Improving Cache Performance

1. Reduce the miss rate,
2. Reduce the miss penalty, or
3. *Reduce the time to hit in the cache.*

CSE 240A

Dean Tullsen

Fast Hit times via Small and Simple Caches

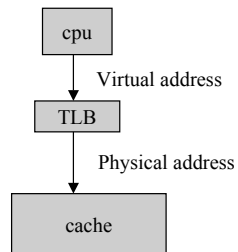
- Why Alpha 21164 has 8KB Instruction and 8KB data cache + 96KB second level cache
- I and D caches used to be typically Direct Mapped, on chip

CSE 240A

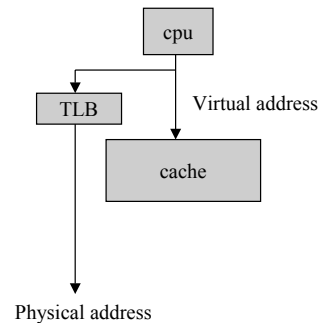
Dean Tullsen

Virtual Cache

- Physical Cache



- Virtual Cache



CSE 240A

Dean Tullsen

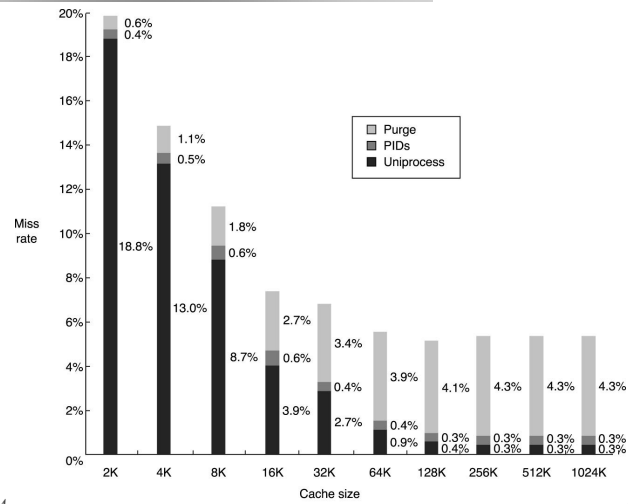
Fast hits by Avoiding Address Translation: Virtual Cache

- Send virtual address to cache? Called *Virtually Addressed Cache* or just *Virtual Cache* vs. *Physical Cache*
 - Every time process is switched logically must flush the cache; 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
 - I/O must interact with cache...
- Solution to aliases
 - HW that guarantees that every cache block has unique physical address
 - SW guarantee : lower n bits must have same address; as long as covers index field & direct mapped, they must be unique; called *page coloring*
- 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

CSE 240A

Dean Tullsen

Avoiding Translation: Process ID impact



CSE 240A

Dean Tullsen

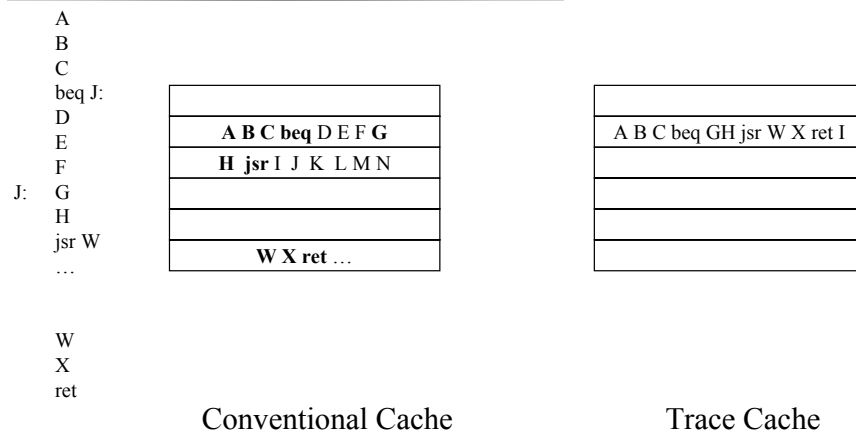
Cache Bandwidth: Trace Caches

- Fetch Bottleneck – Cannot execute instructions faster than you can fetch them into the processor.
- Cannot typically fetch more than about one taken branch per cycle, at best (why? Why one *taken* branch?)
- Trace cache is an instruction cache that stores instructions in dynamic execution order rather than program/address order.

CSE 240A

Dean Tullsen

Trace Cache



Conventional Cache

Trace Cache

CSE 240A

Dean Tullsen

Cache Optimization Summary

Technique	MR	MP	HT	Complexity
Larger Block Size				
Higher Associativity				
Victim Caches				
Pseudo-Associative Caches				
HW Prefetching of Instr/Data				
Compiler Controlled Prefetching				
Compiler Reduce Misses				
Priority to Read Misses				
Early Restart & Critical Word 1st				
Non-Blocking Caches				
Second Level Caches				
Small & Simple Caches				
Avoiding Address Translation				
Trace Cache?				

CSE 240A

Dean Tullsen

Recent Cache Research at UCSD

- Hardware prefetching of complex data structures (e.g., pointer chasing)
- Fetch Target Buffer
 - Let branch predictor run ahead of fetch engine
- Runtime identification of cache conflict misses
- Speculative Precomputation
 - Spawn threads at runtime to calculate addresses of delinquent (problematic) loads and prefetch → creates prefetcher from application code.