Filter Caching for free: The Untapped potential of the Store Buffer → Large + search on every load. reqd.

↓

allow stores to retire under a miss, hiding the latency.

Currently, probing SB, L1, TLB all are done in parallel.

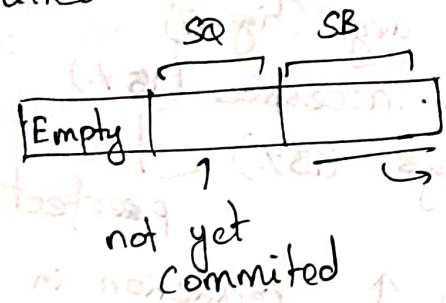
→ This work analyzes SB, and whether a hit in SB can avoid TLB, L1 probes.

(Unified Store queue)

Properties of Store : Free caching, Cheap coherence, free × accurate hit prediction.

Store queue : a) Keep track of original stores' order.
b) Forward data to load instructions that address the same memory location of an uncommitted store.

But, when they are ready to commit, they may be stalled due to cache misses.



To avoid load latency, SQ/SB, L1/TLB are parallely probed. If the address matches a store in the SQ/SB, data is forwarded from the youngest store that matches the address.

SQ/SB Utilization x Hit ratio:

↓

due to low size,

aggressive eviction → low utilization (Show fig 2. here)

↳ ≤ 20% - 62% time on SPEC2006.

↳ ≤ 40% - 85% of the time

Hit ratio - very low except a few

avg < 10% ⇒ Low latency provided by SQ/SB not useful, as it is subsumed by the low hit ratio.

Filter cache: a very small cache between the CPU and L1. few cache lines with high associativity

↓

Low hit rates → probing the filter cache + probing * copying from L1. may be worse than directly accessing L1.

⇓

SQ/SB is a perfect candidate

Optimal SQ/SB - Increase avg (fig 4)

Maybe, reduce L1/TLB accesses (15%) and energy savings. (13%) — perfect case

for most benchmarks, as size ↑, reduction in L1/TLB ↓

56 - Intel - good for most of them

Store Buffer Cache: How to reduce L1/TLB accesses?

↳ ~~reduce~~ delay writebacks so that we'll get hit in SQ/SB. Not good enough - Too much delay leads to lesser availability

↳ One should predict when more capacity is needed.

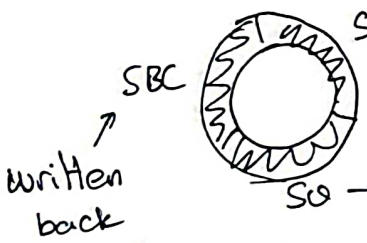
↳ This requires: (i) accurately predicting store-misses
(ii) doing (i) ASAP
(iii) Predict how much to write back.

S/QBC -?

↳ Instead of delay, write back to L1, similar to a traditional SB.

↳ But, keep a copy in SBC.

S/QBC - implemented as a circular queue



Can move from SQ → SB, SB → SBC using pointers
⇒ No extra storage/copying

Store buffer cache synonyms:

Translation from VA → PA reqd.

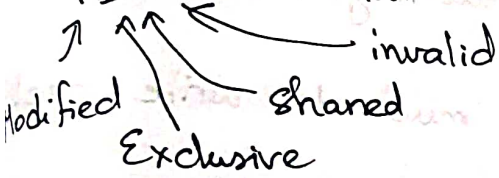
Generally, LQ, SQ, SB, SBC hold both physical & virtual address. But, if PA not found, then perform 1 TLB access.

A load hit on a SBC entry with no PA requires only 1 TLB access. ⇒ for earlier store, and no need for the later load.

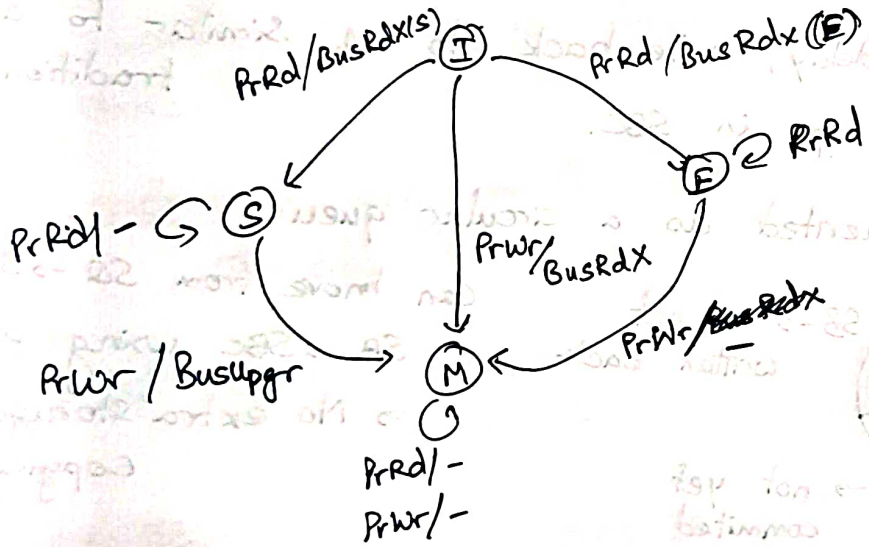
SBC Coherence: Keeping clean copies creates coherence problems.

SBC's data has been written to L1. If some other core has modified the data block, hits in S/OBC can return incoherent values.

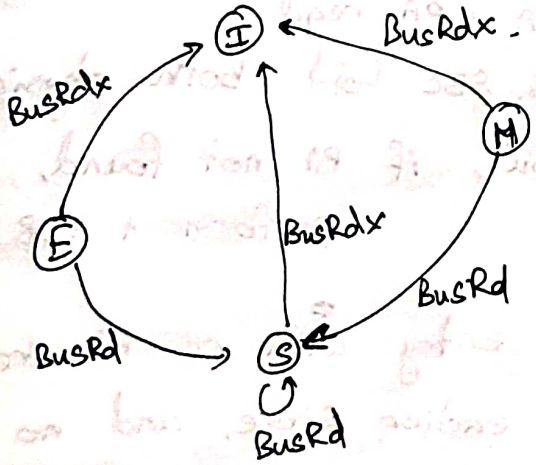
MESI invalidation protocol.



Master core



Bus side



2 naive sols: ⇒ forward any invalidation that reaches L1 and any L1 eviction to the S/OBC.

⇒ we can selectively invalidate individual entries in the SBC portion.

(selectively invalidate ⇒ energy-expensive)

⇒ Bulk flush on any invalidation / eviction

Store written to L1 ⇒ 'M' state.

⇒ L1 invalidations of cache lines of E/S can be ignored.

However if not in M state, we cant say if it affects SBC or not.