Filter Caching for Free: The Untapped Potential of the Store-Buffer

Spectredown (**# 4**)
Prajeeth.S
(190050117)

I. Summary

The authors have identified the potential of the store buffer to act as a filter cache. They have presented a unified store queue/buffer/cache to cache data that has been written back to memory and predict hits to avoid L1/TLB probes and save energy. They demonstrate increase in the Store buffer hit rates, reduction in dynamic energy on the SPEC2006 benchmark without causing avoiding performance degradation and incurring minimal storage overhead.

II. Details

- Standard Store-queue/buffer(SQ/SB) is a circular FIFO consisting of two components. Store queue(SQ) allows stores to retire under cache misses my maintaining order among stores. When the store is committed, it is moved from Store queue(SQ) to the Storage buffer(SB). It follows an aggressive eviction policy where the entry is SB is removed right after writing back to cache.
- Owing to the small size and aggressive eviction policy, SQ/SB utilization and the hit rate is low. So, programs are not able to benefit from the low latency of SQ/SB hits. SQ/SB intrinsically has the probe and copying overheads required for a filter cache.
- A portion of the unified SQ/SB structure is used as a Storage Buffer Cache(SBC), and the combined structure becomes S/QBC. To reduce accesses to L1/TLB, after writing back to cache from SB, the block is moved to the SBC, allowing maximum hit rates until there is a space requirement. In order to predict if the S/QBC gives a hit, a memory dependence predictor based on store-distances is used. Handling the SBC synonym problem(two Virtual addresses mapping to the same physical address) that occur requires at most one TLB access.
- To handle problems due to coherence(where another core can modify data unknown to S/QBC), the SBC entries are flushesd only under restricted conditions depending on the Cache invalidation protocol.
- To avoid flushing too many entries, the authors propose to add extra dirty bits to each cache line that correspond to a ”color” or ”epoch”. On Invalidation/Eviction at a cache line in L1, entries in SBC with the same color as the L1 line are flushed. The others remain unaffected.

III. Strengths

- The proposed SBC requires minimal storage overhead (only 2 or 3 bits per cache line).
- The SBC is a logical portion of the unified SQ/SB. So, moving entries from SB to SBC does not require any copying of data. It suffices to maintain a pointer to mark the boundary between SB and SBC and move it accordingly.
- Using only 2 extra bits(3 colors), the performance improvements nearly match that of the Optimal SQ/SB(which delays write back until there is a space requirement), and hence sufficient for practical usage.

IV. Weaknesses

- Degradation of performance is observed in cases with low read locality and inaccurate memory dependence predictors leading to serialization of S/QBC and L1/TLB probes.
- Performance Improvements are very minimal in case of Parallel Workloads as there are more frequent invalidations and evictions.
- Usage of a finite number of extra dirty bits for color does not eliminate unnecessary flushing from SBC caused due to a relatively old store.

V. Extensions

- The authors have described the coherence protocol for the Total Store Ordering memory model(TSO). This can be extended to weaker memory models.
- The coherence protocol has been described for the simple MESI cache invalidation protocol. This can be extended to more complex MOESI/MESIF protocols.

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

↓

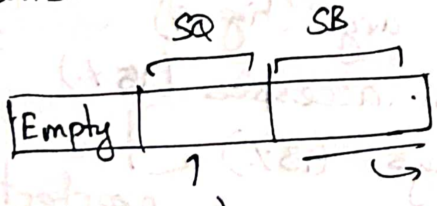
allow stores to retire under a miss, hiding the latency.

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 re$\overset{a}{\wedge}$dy to commit, they may be stalled due to cache misses.



not yet commited

commited, ready to write

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 & 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.

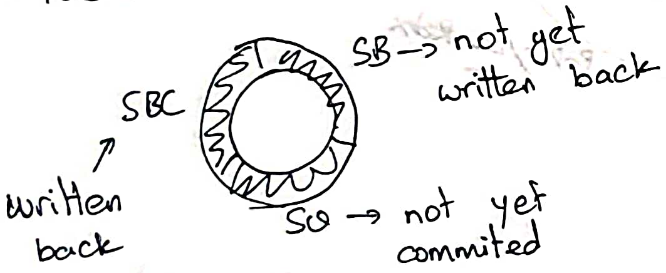
↳ This requires: (i) accurately predicting store-misses
(ii) doing (i) ASAP
(iii) Predict how much to write back.

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.

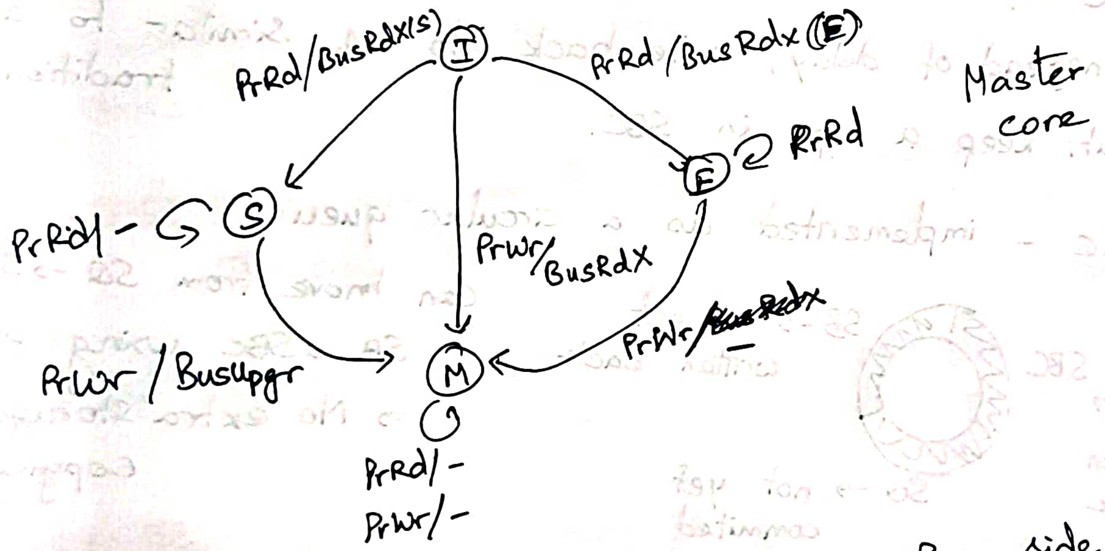
MESI invalidation protocol.

Modified

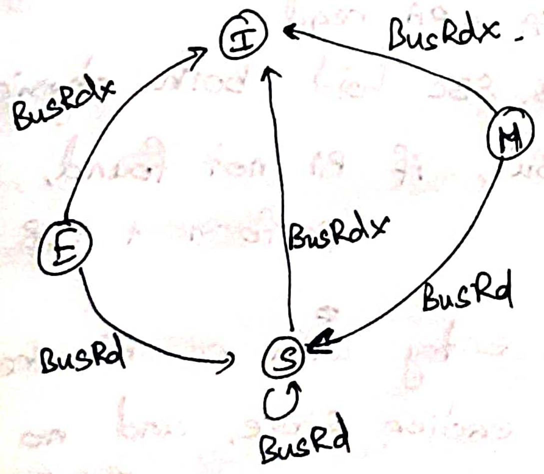
Exclusive

Shared

invalid



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.
$\Downarrow$
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.