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 for L1 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) consists of two components. Store queue(SQ) allows stores to retire under cache misses. It keeps track of the original order of the store instructions. 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 due to low hit rate. 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. Handling the Cache synonym problem that occur requires at most one TLB access.
- To handle problems due to coherence(following MESI invalidation protocol), naive solutions include forwarding any invalidation/eviction at L1 to S/QBC(that flushes out specific entries while being energy-expensive) and bulk flushing all the SBC entries. This can be optimized by performing bulk flushes only under restricted conditions depending on the invalidation protocol.
- To avoid too much of bulk flushing, the authors propose to add extra 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.
- In order to predict if the S/QBC gives a hit, a memory dependence predictor based on store-distances is used.

III. Strengths

- Incurs very minimal storage overheads(only 2 or 3 bits per cache line).
- The incorporation of the SBC to create a new unified S/QBC is a seamless process, as it just involves allocating a portion of the already existing unified SQ/SB, and maintaining a pointer to denote the boundary between SBC and SQ/SB. For the same reason, moving between SQ, SB and SBC is also very efficient.
- The performance improvements using only 3 extra bits(7 colors) per line nearly matches that of the optimal SQ/SB(which delays writing back to cache as much as possible), and hence sufficient for practical usage.

IV. Weaknesses

- Does not work well in cases having applications with very little read locality and inaccurate predictors. For example, SPEC2006 benchmarks libquantum, mcf and milc gives no improvement over the baseline SQ/SB.
- The performance, energy and IPC improvements depends highly on the memory dependence predictors used.
- Due to SBC Coherence, performance improvements are much smaller for parallel workloads as there is more flushing caused due to more frequent invalidations/evictions.
- Although the usage of colors allow much lesser bulk flushing than the naive solution, there is always a possibility of flushing out SBC entries due to a downgrade of a very old store that has already left the SBC.

V. Extensions

- The discussion on coherence is given for the Total Store Order(TSO) memory model. A similar analysis on weaker models can be carried out.
- The restricted conditions mentioned have been for the simple MESI invalidation protocol. This can be extended for more complex MESIF/MOESI 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.

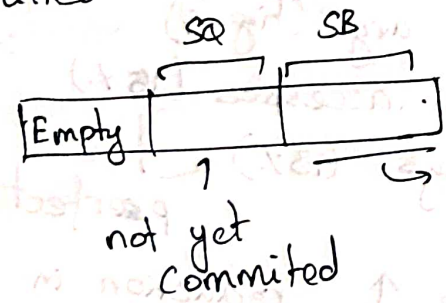
→ This work analyzes SB, and whether a hit in SB can avoid TLB, L1 probes.

(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 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.

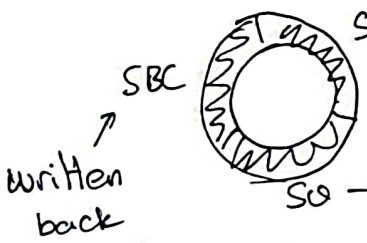
↳ This requires: (i) accurately predicting store-misses
(ii) doing (i) ASAP
(iii) Predict how much to write back.

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.

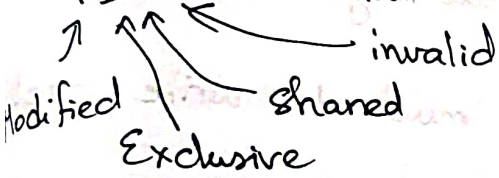
Generally, LQ, SQ, SB, SBC hold both physical & virtual address. But, if PA not found, then perform 1 TLB access.

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.

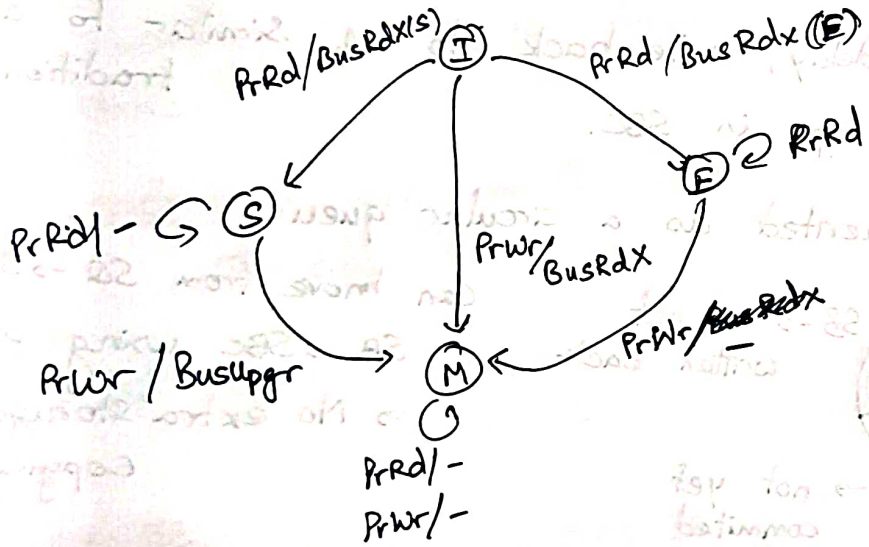
SBC Coherence: Keeping clean copies creates coherence problems.

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.

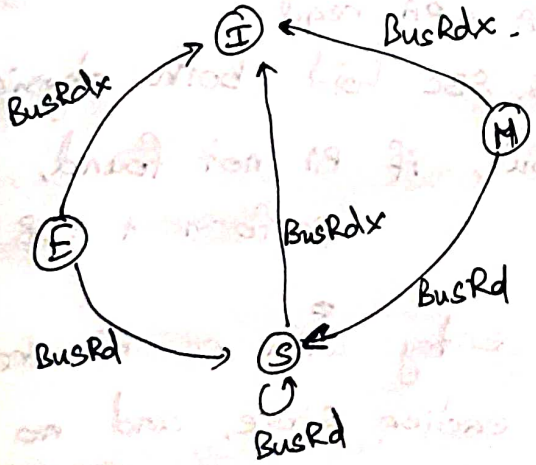
MESI invalidation protocol.



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.