Memory Consistency (II)

Instructor: Josep Torrellas
CS533
Enhancing RC

- Allow an access to partially or fully proceed even though the delay arcs demand that the access be delayed
- Detect and remedy the cases when the early access would result in incorrect behavior.
- How? Re-issue the access to the memory system

- Summary: common case proceeds with high speed and while still preserving correctness
Two Techniques

• Prefetching
• Speculative execution
Prefetching

- Prefetching is classified as:
  - Binding vs non-binding
  - Hardware vs software

- Cache coherent machines can provide non-binding prefetching

- Non-binding prefetching:
  - does not affect the correctness for any consistency model
  - can be used as performance booster

- Can use:
  - for a read: read prefetch
  - for a write: read-exclusive prefetch WHY?

- Bring data into the cache and perform the operation when the memory consistency model allows
What if there is an intervening access

• After a read prefetch a remote processor writes:
  – our copy gets invalidated
  – when the local read is actually issued: it misses

• After a read-exclusive prefetch
  – a remote processor writes: same as above
  – a remote processor reads:
    • our copy loses exclusivity
    • when the local write is issued: miss
Implementation

• Assume a processor with LD and ST buffers
• Local access kept in buffer, it is delayed until it is correct to do it
• Hardware automatically issued:
  – Read prefetch: for reads in the buffer
  – Read-exclusive pref: for writes (and RMW) in the buffer
• Prefetches are buffered in a special prefetch buffer. They are retired as quickly as the bandwidth of the memory system allows
• Prefetch first checks the cache. If data there in the right state, then prefetch is discarded
• Prefetch response is placed into the cache
Implementation

- If processor references the line before the prefetch has arrived, no additional request is issued to the memory system (combining)
Issues

• Prefetching
  – needs lock up free caches
  – causes additional cache accesses (double)
  – but: only if the request cannot proceed past the LD/ST buffer because of memory consistency problems
  – best with out-of-order execution and aggressive branch prediction
Examples

- Examples (assume that the branch prediction takes the path that assumes such succeeds)

  lock L (miss)         Lock L (miss)
  write A (miss)       read C (miss)
  write B (miss)       read D (hit)
  unlock L (hit)       read E[D]  (miss)
  unlock L (hit)

EX1: SC: 301, RC: 202, with pref (SC or RC): 103
EX2: SC: 302, RC: 203, with pref: 203 SC and 202 RC

note: the E[D] not allow to perform until read to C completes (under SC) or lock access completes (under RC)
Summary

• Pref under RC does not help when out-of-order consumption of return values is important to allow the processor to proceed efficiently.
Speculative Execution

• Allow the processor to consume return values ooo regardless of the consistency constraints

• Goal: combination of:
  – prefetching for stores
  – speculative execution for loads
Speculative Execution

• Consider access u (long latency) followed by v (a load)
• Assume that the consistency model requires v to be delayed until u completes
• Speculative execution: the processor obtains or assumes a return value for v before u completes, and proceeds
• When u completes:
  – if current value value of v is as expected, speculation was successful
  – if current value is different: throw out the computation that depended on the value of v and re-execute
Requirements

• Speculation mechanism: obtain the speculated value
• Detection mechanism:
• Correction mechanism: to repeat the computation if mis-speckulated
Mechanisms

• Speculation mechanism: Perform the access:
  – if cache hit: return immediately
  – if miss: takes longer

• Detection mechanism:
  – Naïve: repeat the access when legal and compare the value
  – Better: keep the data in the cache and monitor if you received a coherence transaction for it
  – Result: cache accessed once rather than twice (as pref)
  – Coherence transactions:
    • invalidation
    • but: false sharing and same-value update
    • But what if cache displacement? Assume that the value is stale
Mechanisms

• Correction mechanism:
  – Discard the computation that depended on the speculated value and repeat the access and computation
  – Same mechanism as processors with branch prediction and ability to execute instructions past unresolved branches
  – Branches: if wrong prediction: instructions past branch are discarded
  – If mis-speculation of the load: instructions past the load are discarded and the load is retried
Summary of Spec Exec

- Allows out of order consumption of speculated values

- In the example 2: value of D is allowed to be used to access E[D]
- Both RC and SC complete in 104 cycles
- Overall: LDs are issued as soon as their address is known
- But: hardware coherence is needed to detect mis-speculations
Conclusion

- Speculative execution is naturally supported by ooo processors
- With speculative execution for load and prefetches for writes, all memory consistency models deliver high performance
- SC gets closer to RC