EECS 570

Lecture 11

Dir. Optimization & COMA

Winter 2022

Prof. Yatin Manerkar

http://www.eecs.umich.edu/courses/eecs570/

Slides developed in part by Profs. Adve, Falsafi, Hill, Lebeck, Martin, Narayanasamy, Nowatzyk, Reinhardt, Roth, Smith, Singh, and Wenisch.
A Few Clarifications

- Directory organization
  - Centralised directory
  - Distributed directory
  - Directory cache
    - Can be distributed too

- Sharer bit vectors vs limited pointers
  - Bit vectors can be coarse
    - 0 1 1 0 0 0 0 1

- Differences between snooping and directory protocols
  - See primer for further details
Designing a Directory Protocol: Nomenclature

• Local Node (L)
  □ Node initiating the transaction we care about

• Home Node (H)
  □ Node where directory/main memory for the block lives

• Remote Node (R)
  □ Any other node that participates in the transaction
Read Transaction

- L has a cache miss on a load instruction

Diagram:
- L (L) has a cache miss on a load instruction
- H (H)
- 1: Get-S
- 2: Data
4-hop Read Transaction

- L has a cache miss on a load instruction
  - Block was previously in modified state at R

Diagram:
1. Get-S
2. Recall
3. Data
4. Data

State: S
Shr: L, R
3-hop Read Transaction

- L has a cache miss on a load instruction
  - Block was previously in modified state at R

![Diagram showing the 3-hop read transaction process]

1: Get-S
2: Fwd-Get-S
3: Data

State: S
Shr: L, R

L
H
R
An Example Race: Writeback & Read

- L has dirty copy, wants to write back to H
- R concurrently sends a read to H

If you think deeply about it:
Can optimize away SI^A & Put-Ack!
L and H each know the race happened, don’t need more msgs.
Store-Store Race

• Line is invalid, both L and R race to obtain write permission

Race! Stall for Data, do 1 store, then Fwd to R

Fwd-Get-M to L; New Owner: R

1: Get-M

Get-M

3: IMAD

5: IMAD

7: 4: Data [ack=0]

6: Fwd-Get-M

8: Data [ack=0]
Another Race

- L evicts dirty copy, R concurrently seeks write permission

Race! Put-M floating around! Wait till its gone…

Put-M from NonOwner: Race! L waiting to ensure Put-M gone…

1: Put-M
2: Get-M
3: Fwd-Get-M
4: Data [ack=0]
5: 
6: Put-Ack
Design Principles

• Think of sending and receiving messages as separate events
• At each “step”, consider what new requests can occur
  □ E.g., can a new writeback overtake an older one?
• Two messages traversing same direction implies a race
  □ Need to consider both delivery orders
    ▪ Usually results in a “branch” in coherence FSM to handle both orderings
  □ Need to make sure messages can’t stick around “lost”
    ▪ Every request needs an ack; extra states to clean up messages
  □ Often, only one node knows how a race resolves
    ▪ Might need to send messages to tell others what to do
CC Protocol Scorecard

• Does the protocol use negative acknowledgments (retries)?
• Is the number of active messages (sent but unprocessed) for one transaction bounded?
• Does the protocol require clean eviction notifications?
• How/when is the directory accessed during transaction?
• How many lanes are needed to avoid deadlocks?
NACKs in a CC Protocol

- Issues: Livelock, Starvation, Fairness
- NACKs as a flow control method (“home node is busy”)
  - Really bad idea...
- NACKs as a consequence of protocol interaction...

1: Put-M
2: Get-S
3: Fwd-Get-S
4: Race!
5: Get-S NACK
6: Race!

Final State: S
No need to Ack

Race! Put-M & Fwd-Get-S
Bounded # Msgs / Transaction

- Scalability issue: how much queue space is needed
- Coarse-vector vs. cruise-missile invalidation
Frequency of Directory Updates

• How to deal with transient states?
  - Keep it in the directory: unlimited concurrency
  - Keep it in a pending transaction buffer (e.g., transaction state register file): faster, but limits pending transactions

• Occupancy free: Upon receiving an unsolicited request, can directory determine final state solely from current state?
Required # of lanes

- Need at least 2:
  - More may be needed by I/O, complex forwarding
  - How to assign lane to message type?
    - Secondary (forced) requests must not be blocked by new requests
    - Replies (completing a pending transaction) must not be blocked by new requests
Some more guidelines

- All messages should be ack’d (requests elicit replies)
- Maximum number of potential concurrent messages for one transaction should be small and constant (i.e., independent of number of nodes in system)
- Use context information to avoid NACKs
Optimizing coherence protocols

- Read A (miss)
- Get-S A
- Recall A
- Data A
- Data A
- Read latency
Prefetching

L

Prefetch A

Get-S A

H

Recall A

Data A

Read A (miss)

Data A

R

Read latency

EECS 570
3-hop reads

Read $A$ (miss)

Read latency

Get-$S$ $A$

Fwd-Get-$S$ $A$

Data $A$

Data $A$
3-hop writes

L → H → R

Store A (miss)

Get-M A

Data [ack=x]

Inv-Ack A

Invalidate A

Store latency
Migratory Sharing

Node 1
Read X
Write X

Node 2

Node 3
Read X
Write X

• Each Read/Write pair results in read miss + upgrade miss

• Coherence FSM can detect this pattern
  - Detect via back-to-back read-upgrade sequences
  - Transition to “migratory M” state
  - Upon a read, invalidate current copy, pass in “mig E” state
Producer Consumer Sharing

- Upon read miss, downgrade instead of invalidate
  - Detect because there are 2+ readers between writes
  - O state can help reduce number of writebacks

- More sophisticated optimizations
  - Keep track of prior readers
  - Forward data to all readers upon downgrade
Shortcomings of Protocol Optimizations

- Optimizations built directly into coherence state machine
  - Complex! Adds more transitions, races
  - Hard to verify even basic protocols
  - Each optimization contributes to state explosion
  - Can target only simple sharing patterns
  - Can learn only one pattern per address at a time
Cache Only Memory Architecture (COMA)
Big Picture

- Centralized shared memory
- Uniform access

- Distributed Shared memory
- Non-uniform access latency

- No notion of “home” node; data moves to wherever it is needed
- Individual memories behave like caches
Cache Only Memory Architecture (COMA)

- Make all memory available for migration/replication
- All memory is DRAM cache called attraction memory

- Example systems
  - Data Diffusion Machine
  - KSR-1 (hierarchical snooping via ring interconnects)
  - Flat COMA (fixed home node for directory, but not data)

- Key questions:
  - How to find data?
  - How to deal with replacements?
  - Memory overhead
COMA Alternatives

• Flat-COMA
  □ Blocks (data) are free to migrate
  □ Fixed directory location (home node) for a physical address

• Simple-COMA
  □ Allocation managed by OS and done at page granularity

• Reactive-NUMA
  □ Switches between Simple-COMA and NUMA with remote cache on per-page basis