DIRECTORY COHERENCE

Mahdi Nazm Bojnordi

Assistant Professor
School of Computing
University of Utah
Overview

- Upcoming deadline
  - Tonight: project proposal

- This lecture
  - Snooping wrap-up
  - Directory coherence
  - Implementation challenges
  - Token-based coherence protocol
Recall: Cache Coherence

- Definition of coherence
  - Write propagation
    - Write are visible to other processors
  - Write serialization
    - All write to the same location are seen in the same order by all processes
Implementation Challenges

- MSI implementation
  - Stable States

[Vantrease’11]
Implementation Challenges

- MSI implementation
  - Stable States
  - Busy states

[Vantrease’11]
Implementation Challenges

- MSI implementation
  - Stable States
  - Busy states
  - Races

Unexpected events from concurrent requests to same block

[Vantrease’11]
Cache Coherence Complexity

- A broadcast snooping bus (L2 MOETSI)

[Lepak’03]
Implementation Tradeoffs

- Reduce unnecessary invalidates and transfers of blocks
  - Optimize the protocol with more states and prediction mechanisms

- Adding more states and optimizations
  - Difficult to design and verify
    - lead to more cases to take care of
    - race conditions
  - Gained benefit may be less than costs (diminishing returns)
Coherence Cache Miss

- **Recall**: cache miss classification
  - Cold (compulsory): first access to block
  - Capacity: due to limited capacity
  - Conflict: many blocks are mapped to the same set

- **New class**: misses due to sharing
  - True vs. false sharing
Summary of Snooping Protocols

□ Advantages
  ▪ Short miss latency
  ▪ Shared bus provides global point of serialization
  ▪ Simple implementation based on buses in uniprocessors

□ Disadvantages
  ▪ Must broadcast messages to preserve the order
  ▪ The global point of serialization is not scalable
    ▪ It needs a virtual bus (or a totally-ordered interconnect)
Scalable Coherence Protocols

- **Problem:** shared interconnect is not scalable
- **Solution:** make explicit requests for blocks

- **Directory-based coherence:** every cache block has additional information
  - To track of copies of cached blocks and their states
  - To track ownership for each block
  - To coordinate invalidation appropriately
Directory Information

- P+1 additional bits for every cache block
  - One bit used to indicate the block is in each cache
  - One exclusive bit to indicate the cache has the only copy (can update without notifying others)
- On a read, set the cache’s bit and arrange the supply of data
- On a write, invalidate all caches that have the block and reset their bits

P=4

\[ \text{P=4} \quad \text{E} \quad \text{Cache Block} \]

How to organize directory information?
Directory Organization

- Example: central directory for P processors
  - For each cache block in memory
    - p presence bits, 1 dirty bit
  - For each cache block in cache
    - 1 valid bit, and 1 dirty (owner) bit

![Diagram of directory organization](image-url)
Three states (similar to snoopy protocol)

- **Shared**: more than one processors have data, memory up-to-date
- **Uncached**: no processor has it; not valid in any cache
- **Exclusive**: one processor has data; memory out-of-date

Basic terminology

- **Local node**, where a request originates
- **Home node**, where the memory location of an address resides
- **Remote node**, has copy of a cache block, whether exclusive or shared
P0 reads a cache location

1. Read

2. DatEx (DatShr)
ReadEx Request

- Avoid roundtrip to home by sending data directly from owner

[Culler/Singh]
What are the challenges?

[Culler/Singh]
Design Challenges

- **Fairness**: which requester is preferred on a conflict?
  - Consider distance and delivery order of interconnect

- **Race condition**: how to keep the proper sequence
  - NACK requests to busy blocks (pending invalidate)
    - Original requestor retries
  - Queuing requests and granting in sequence
Summary of Directory Protocols

- **Advantages**
  - Does not require broadcast to all caches
  - Exactly as scalable as interconnect and directory storage (much more scalable than bus)

- **Disadvantages**
  - Adds *indirection* to miss latency (critical path)
    - request → directory → memory
  - Requires extra storage space to track directory states
  - Protocols and race conditions are more complex
Avoid Indirection

- Can we get the best of both snooping and directory protocols?
  - Direct cache-to-cache misses (broadcast is ok)
  - What if unordered interconnect (e.g., mesh) was used?
An Example Problem

- $P_0$ issues a request to write (delayed to $P_2$)
- $P_1$ issues a request to read

Diagram:
- $P_0$ (No Copy) → $P_1$ (No Copy) → $P_2$ (Read/Write)
- $P_0$ (No Copy) → $P_1$ (Ack)
- $P_1$ (No Copy) → $P_2$ (Read/Write)
- Request to write delayed in interconnect

Flow:
1. $P_0$ requests write
2. $P_1$ acknowledges
3. $P_1$ requests read
An Example Problem

- $P_0$ responds with data to $P_1$

- $P_2$ responds with data to $P_1$
An Example Problem

- $P_0$’s delayed request arrives at $P_2$
An Example Problem

- $P_2$ responds to $P_0$
Problem: $P_0$ and $P_1$ are in inconsistent states. Locally “correct” operation, globally inconsistent.
Token Coherence

- $P_0$ issues a request to write (delayed to $P_2$)
- $P_1$ issues a request to read

[Martin’03]
Token Coherence

• $P_2$ responds with data to $P_1$

[Martin’03]
Token Coherence

- $P_0$’s delayed request arrives at $P_2$

[Martin’03]
Token Coherence

- $P_2$ responds to $P_0$

[Martin’03]
Token Coherence

T=15(R) \hspace{1cm} T=1(R) \hspace{1cm} T=0

\begin{align*}
P_0 & \hspace{1cm} P_1 & \hspace{1cm} P_2
\end{align*}

Now what? (P_0 wants all tokens)

[Martin’03]
Token Coherence

- $P_0$ reissues request
- $P_1$ responds with a token

[Martin’03]
Token Coherence

One final issue: What about starvation?

- $P_0$’s request completed

$P_0$ (T=16 (R/W))  $P_1$ (T=0)  $P_2$ (T=0)

[Martin’03]