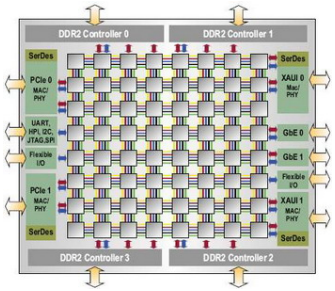
Advanced Computer Architecture

Advanced Coherence

Fall 2016



Pejman Lotfi-Kamran

Adapted from slides originally developed by Profs. Hill, Hoe, Falsafi and Wenisch of CMU, EPFL, Michigan, Wisconsin

Where Are We?

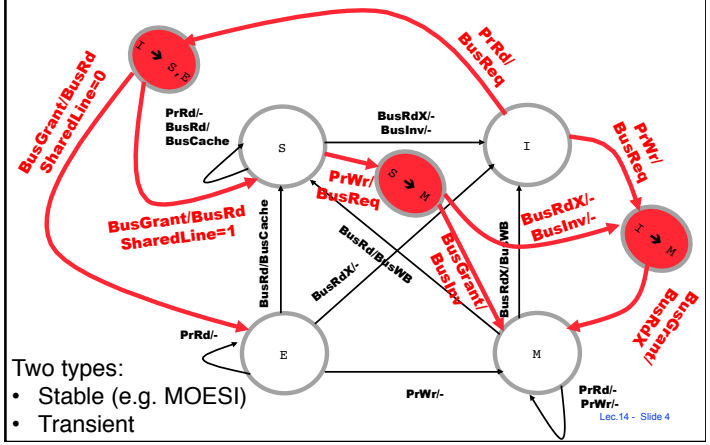
Fr	Sa	Su	Mo	Tu
	27-Shahrivar		29-Shahrivar	
	3-Mehr		5-Mehr	
	10-Mehr		12-Mehr	
	17-Mehr		19-Mehr	
	24-Mehr		26-Mehr	
	1-Aban		3-Aban	
	8-Aban		10-Aban	
	15-Aban		17-Aban	
	22-Aban		24-Aban	
	29-Aban		1-Azar	
	6-Azar		8-Azar	
	13-Azar		15-Azar	
	20-Azar		22-Azar	
	27-Azar		29-Azar	
	4-Dey		6-Dey	

- This Lecture
  - Advanced Coherence
    - ▲Efficient snooping
    - ▲Directories
    - ▲Optimization
- Next Lecture:
  - Consistency

Non-Atomic State Transitions

- Operations involve multiple actions
  - Look up cache tags
  - Bus arbitration
  - Check for writeback
  - Even if bus is atomic, overall set of actions is not
  - Race conditions among multiple operations
- Suppose P1 and P2 attempt to write cached block A
  - Each decides to issue BusUpgr to allow S –> M
- Issues
  - Handle requests for other blocks while waiting to acquire bus
  - Must handle requests for this block A

Non-Atomicity ➔Transient States (from 4 to 7 states)



Two types:
- Stable (e.g. MOESI)
- Transient

Multi-level Cache Hierarchies

- How to snoop with multi-level caches?
  - independent bus snooping at every level?
  - maintain cache inclusion
- Requirements for Inclusion
  - data in higher-level is subset of data in lower-level
  - modified in higher-level => marked modified in lower-level
- Now only need to snoop lowest-level cache
  - If L2 says not present (modified), then not so in L1
- Is inclusion automatically preserved
  - Replacements: all higher-level misses go to lower level
  - Modifications

Violations of Inclusion

- The two caches (L1, L2) may choose to replace different block

  Example: Local LRU not sufficient

  Assume that L1 and L2 hold two and three blocks and both use local LRU

  Processor references: 1, 2, 1, 3, 1, 4

  Final contents of L1: 1, 4

  L1 misses: 1, 2, 3, 4

  Final contents of L2: 2, 3, 4, but not 1

Violations of Inclusion

- Split higher-level caches
  - instruction, data blocks go in different caches at L1, but collide in L2
- Differences in Associativity
  - What if L1 is set-associative and L2 is direct-mapped?
- Differences in block size
  - Blocks in two L1 sets may both map to same L2 set
- *But* a common case works automatically
  - L1 direct-mapped, fewer sets than in L2, and block size same

Inclusion: to have or not to have

- Most common inclusion solution
  - Ensure L2 holds superset of L1I and L1D
  - On L2 replacement or coherence request that must source data or invalidate, forward actions to L1 caches
  - Can maintain bits in L2 cache to filter some actions from forwarding
  - virtual L1 / physical [Wang, et al., ASPLOS87]
- But
  - Restricted associativity in unified L2 can limit blocks in split L1's
  - Not that hard to always snoop L1's *e.g., on-chip)
- Thus, many new designs don't maintain inclusion

Summary

- Lots of possibilities with FSMs
  - Invalidate/upgrade protocols
  - What do we do with the 4th state (Owner-based, Exclusive-based protocols)
- Think about implementation too
  - 4 states become many more states. Why?
  - Multiple cache levels
  - Inclusion
  - Serialization and ordering

Roadmap

- ➔ Snoopy optimization
- Directory protocols
- Protocol optimization
- Directory optimization

Power is Becoming Important

- Architecture is a science of tradeoffs
- Thus far:

  Performance vs. Cost vs. Complexity
- Past (past) Decade or so:

  vs. ***Power***
- Where?
  - **Mobile Devices**
  - **Desktops/Servers** ← **Our Focus**

Power-Aware Servers

- **Revisit the design of SMP servers**
  - 2 or more CPUs per machine
  - Snoop coherence-based
- **Why?**
  - File, web, databases, your typical desktop
  - Cost effective too
- This work - a first step:

  Power-Aware Snoopy-Coherence

Power-Aware Snoop-Coherence

- **Conventional**
  - *All* L2 caches snoop *all* memory traffic
  - Power expended by *all* on *any* memory access
- **Jetty-Enhanced**
  - Tiny structure on L2-backside
  - Filters most "would-be-misses"
  - Less power expended on most snoop misses
  - No changes to protocol necessary
  - No performance loss

Roadmap

- Why Power is a Concern for Servers?
- Snoopy-Coherence Basics
- An Opportunity for Reducing Power
- JETTY
- Results
- Summary

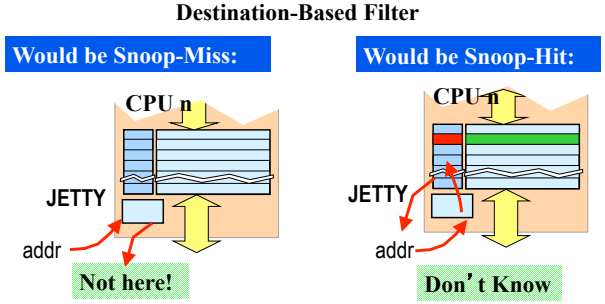
Why is Power Important?

Power Could Ultimately Limit Performance

- Power Demands have been increasing
- Deliver Energy to and on chip
- Dissipate Heat
- Limit:
  - Amount of resources & frequency
  - Feasibility
- Cooling a solution: Cost & Integration?

Reducing Power Demands is much more convenient

JETTY: A Would be Snoop-Miss Filter Built in IBM BlueGene [Moshovos' 01]



Detect most misses using fewer resources

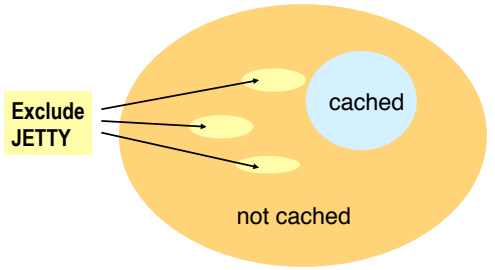
Imprecise: May filter a would-be miss
Never filters snoop-hits

Potential for Savings Exist

- **Most Snoops miss**
  - 91% AVG
- **Many L2 accesses are due to Snoop Misses**
  - 55% AVG
- **Sizeable Potential Power Savings:**
  - 20% - 50% of total L2 power

Exclude-Jetty

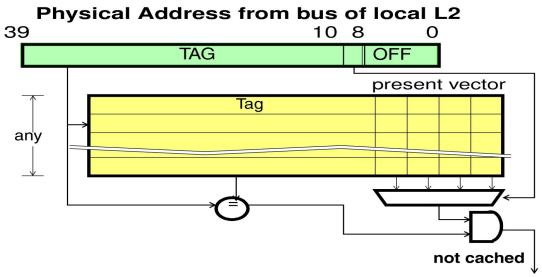
- Subset of what is not cached



How? Cache recent snoop-misses locally

Exclude-Jetty

- Subset of what you don't have

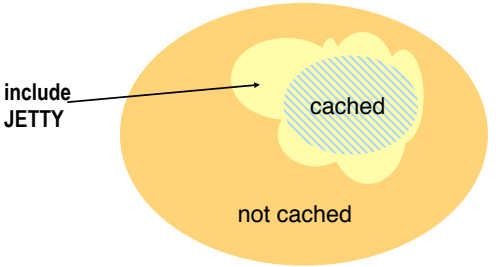


Works well for producer-consumer

Other nodes see the traffic and they don't care

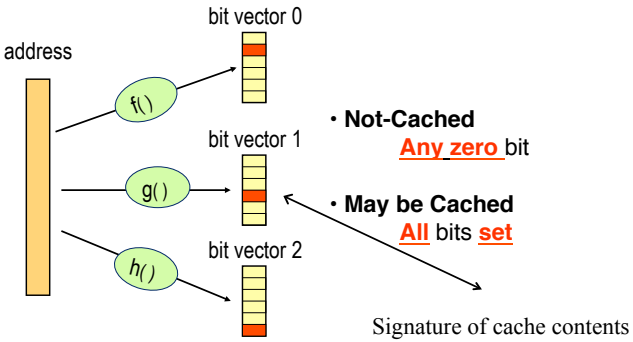
Include-Jetty

- Superset of what is cached



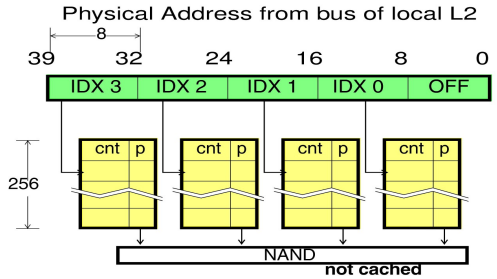
How? Well...

Include-Jetty (A Counting Bloom Filter)



Include-Jetty

- Superset of what you have



Partial overlapping indexes worked better

Hybrid-Jetty

- Some cases Exclude-J works well
- Some other Include-J is better
- Combine
  - Access in parallel on snoop
  - Allocation
    - IJ always
    - If IJ fails to filter then to EJ
    - EJ coverage increases

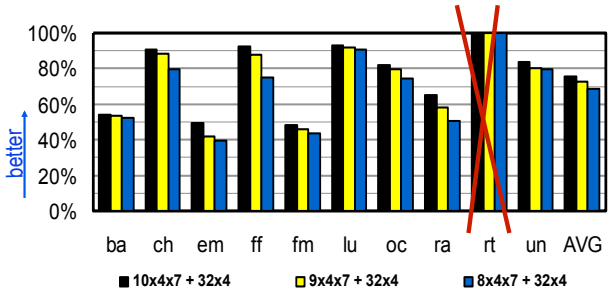
Latency?

- Jetty may increase snoop-response time
- Can only be determined on a design by design basis
- **Largest Jetty:**
  - **Five 32x32 bit register files**

Results

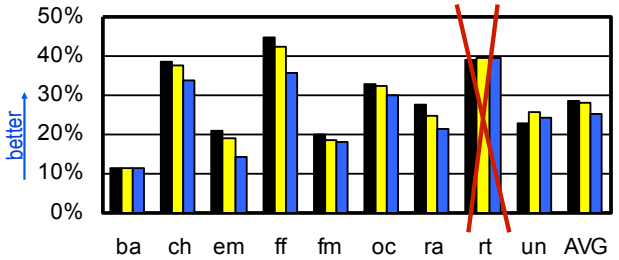
- Used SPLASH-II
  - Scientific applications
  - "Large" Datasets
    - ▲e.g., 4-80Megs of main memory allocated
    - ▲Access Counts: 60M-1.7B
  - 4-way SMP, MOESI
  - 1M direct-mapped L2, 64b 32b subblocks
  - 32k direct-mapped L1, 32b blocks
- Coverage & Power (analytical model)

Coverage: Hybrid-Jetty



- Can capture 74% of all snoop-misses

Power-Savings



- 28% of overall L2 power

Summary

- Much energy is wasted in snoops
  - Save energy by through filters
- HPCA'01 paper
- Followup in an Intel Patent (cites Jetty)
- Implemented in IBM BlueGene/P

Roadmap

- Snoopy optimization
- → Directory protocols
- Protocol optimization
- Directory optimization

Connecting the processors

Connecting multiple chips

- Snoopy bus-based coherence
- Works for connecting chips
  - Pin latency high (must go through I/O pads to reach pins)
  - Pin b/w reasonable
  - Buses are ideal for a small number of processors

On chip

- Can connect using bus
- But, buses become a bottleneck
- Too slow, too much energy

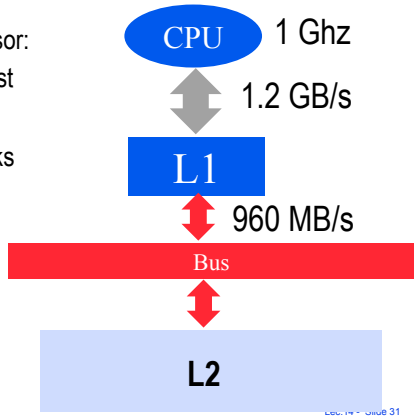
Bus on Chip

Assume a 1 Ghz processor:

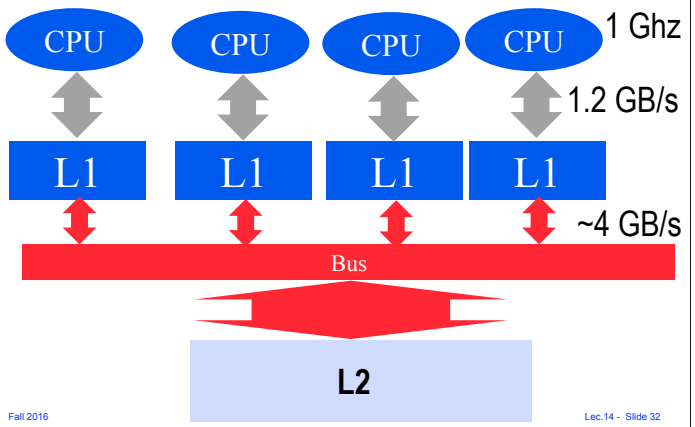
- Scalar, RISC, 30% ld/st

Assume misses 10%

Assume 32 byte L1 blocks



B/W demand goes up quickly!

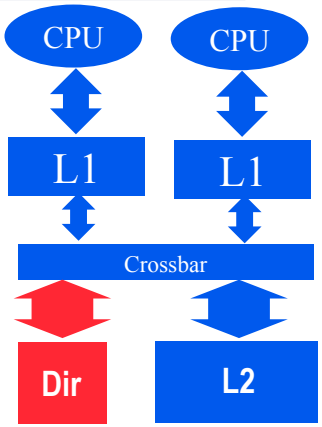


Directory-Based Cache Coherence

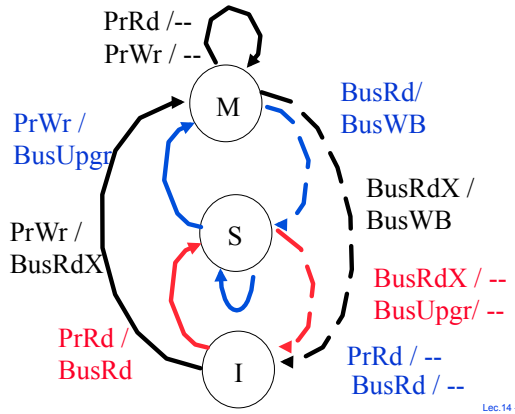
- Avoid broadcast request to all nodes on a miss
  - traffic
  - time
- Maintain directory of which nodes have cached copies of the block (directory controller + directory state)
- On a miss, send message to directory
  - communication assist
- Directory determines what (if any) protocol action is required
  - e.g., invalidation
- Directory waits for protocol actions to finish and then responds to the original request

Directories

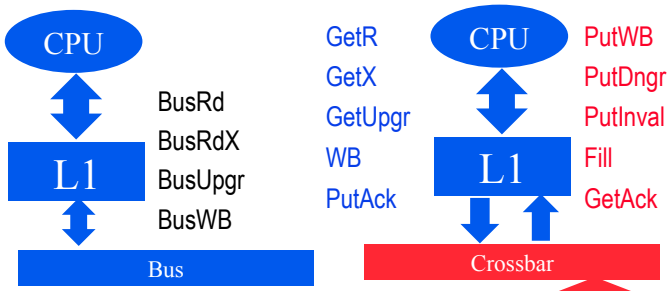
- Keep track of sharers in L1's
- Upon access:
  - Consult with the directory
  - Find out if there are other copies
  - If so, get a coherent copy
  - If not, mark the directory to indicate your copy



Review: L1's snoopy MSI



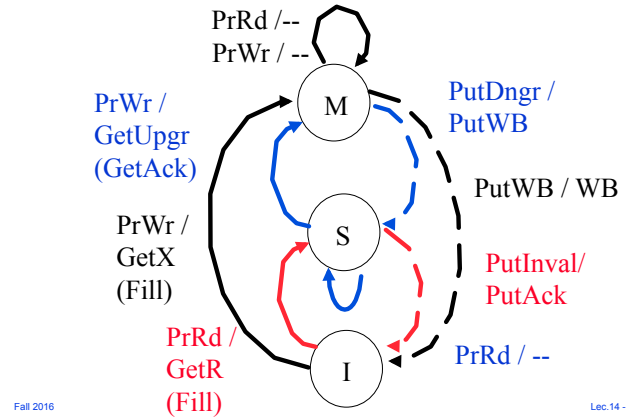
But, the interface is a bit different



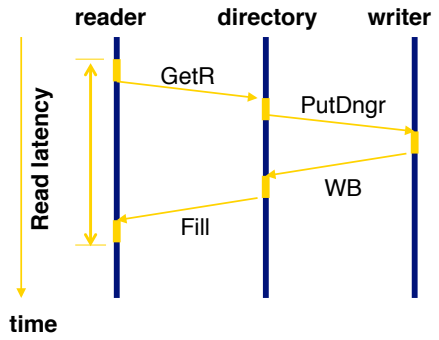
In the bus, transactions seen by all

In the crossbar, point-to-point messages between L1's and directory

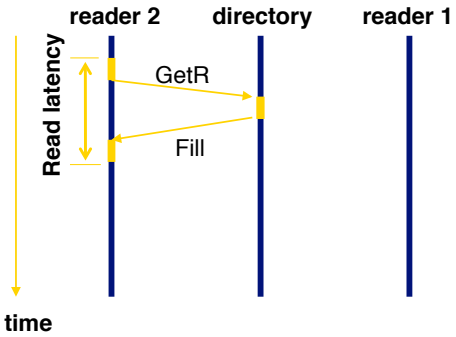
L1's MSI with Directories



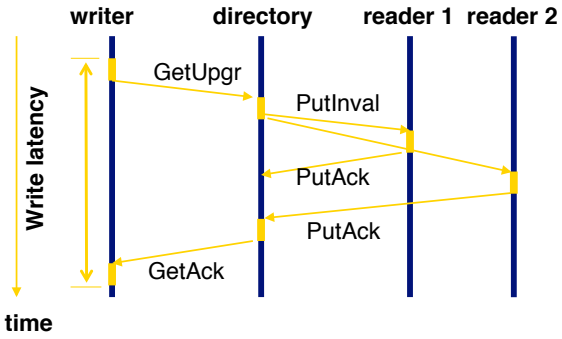
Example I ➔ S with Writer



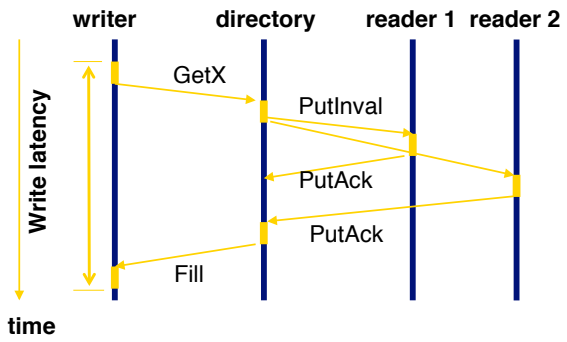
Example I ➔ S with Reader



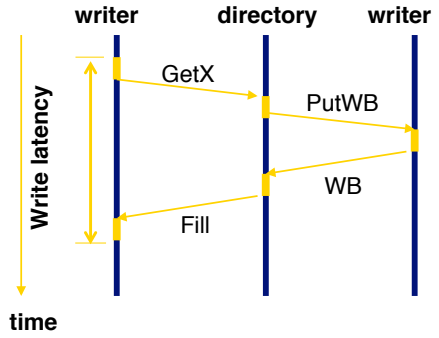
Example S ➔ M



Example I ➔ M with Readers



Example I ➔ M with Writer



Cache Block Fill: Just like Snoopy Bus Systems

- Directory says which L1 has a copy
- Block data can come from other L1's or L2
  - L2 is closer (crossbar already traversed) so check in L2 first
  - If not, can get from next level (L3 or off-chip)
  - or get from another L1
    - Need cache controller support
    - A "copyback" for a block in the S state!
  - Depends on L2 inclusion

A Note on L2 Inclusion

- If L2 inclusive
  - On a WB, update copy in L2
  - Upon subsequent reads, data comes from L2
- If L2 non-inclusive
  - A read might find other L1 copies
  - Can get data from L1
  - Or go to the next level

No snoop, but centralize around directory

- No snoop because of the network is not a point of serialization
- But, make directory the point of serialization
  - ▲ Requests for block A are serialized at directory
  - ▲ Requests across blocks (e.g., for block B) can overlap
- For example
  - ● Read miss for A blocks until read completes
  - ● Read miss for A from another CPU blocks while the first is pending
  - ● Read miss for B proceeds while A is pending