Scaling Cache Coherence

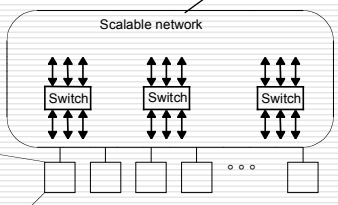
Zeljko Zilic
McConnell Engineering Building
Room 546

McGill

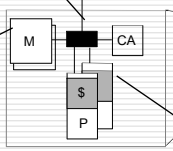
Scalable CC - Context

Realizing Pgm Models through net transaction protocols
- efficient node-to-net interface
- interprets transactions

Scalable Networks
- many simultaneous transactions



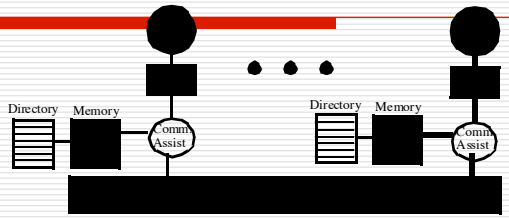
Scalable distributed memory



Caches naturally replicate data
- coherence through bus snooping protocols
- consistency

Need cache coherence protocols that scale!
- **In general, no broadcast or single point of order**

Generic Solution: Directories



- Maintain state vector explicitly
  - Associated with memory block (cache line)
  - Records state of block in each cache
- On miss, communicate with directory
  - Determine location of cached copies
  - Determine action to take
  - Conduct protocol to maintain coherence

Cache Coherent System Must:

- Provide set of states, state transition diagram, and actions
- Manage coherence protocol
  - (0) Determine when to invoke coherence protocol
  - (a) Find info about state of block in other caches to determine action
    - Whether need to communicate with other cached copies
  - (b) Locate the other copies
  - (c) Communicate with those copies (inval/update)
- (0) is done the same way on all systems
  - State of the line is maintained in the cache
  - Protocol is invoked if an "access fault" occurs on the line
- Different approaches distinguished by (a) to (c)

Bus-based Coherence

- All of (a), (b), (c) done through broadcast on bus
  - Faulting processor sends out a "search"
  - Others respond to the search probe and take necessary action
- Could do it in scalable network too
  - Broadcast to all processors, and let them respond
- Conceptually simple, but broadcast doesn't scale with p
  - On bus, bus bandwidth doesn't scale
  - Nn scalable network, every fault leads to at least p network transactions
- Scalable coherence:
  - Can have same cache states and state transition diagram
  - Different mechanisms to manage protocol

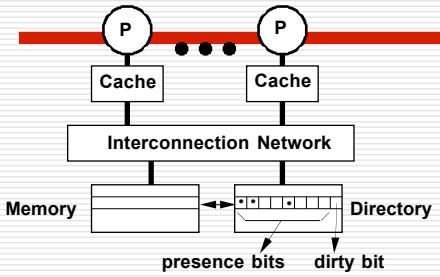
A Try: Hierarchical Snooping

- Extend snooping approach: hierarchy of broadcast media
  - Tree of buses or rings (KSR-1)
  - Processors are in the bus- or ring-based multiprocessors at leaves
  - Parents and children connected by two-way snoopy interfaces
    - Snoop both buses and propagate relevant transactions
  - Main memory may be centralized at root or distributed among leaves
- Issues (a) - (c) handled similarly to bus, but not full broadcast
  - Faulting processor sends out "search" bus transaction on its bus
  - Propagates up and down hierarchy based on snoop results
- Problems:
  - High latency: multiple levels, and snoop/lookup at every level
  - Bandwidth bottleneck at root
- Not popular today

Scalable Approach: Directories

- Every memory block has associated directory information
  - Keeps track of copies of cached blocks and their states
  - On a miss, find directory entry, look it up, and communicate only with the nodes that have copies if necessary
  - In scalable networks, communication with directory and copies is through network transactions
- Many alternatives for organizing directory information

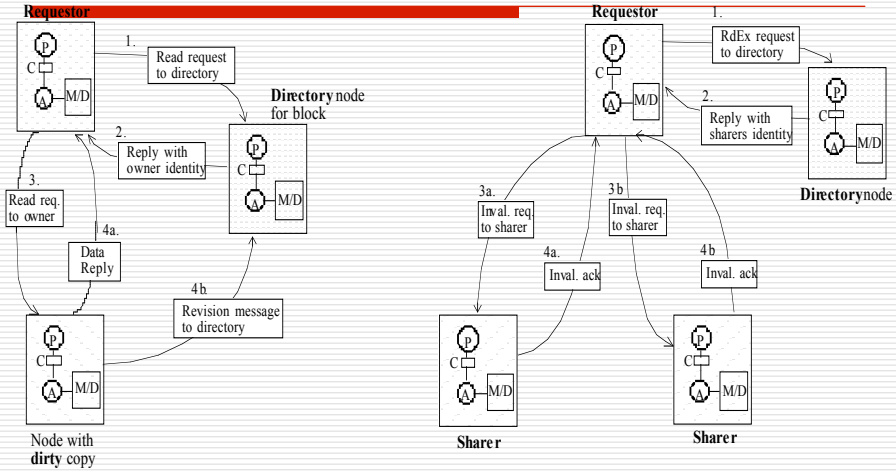
Basic Operation of Directory



- k processors.
- With each cache-block in memory: k presence-bits, 1 dirty-bit
- With each cache-block in cache: 1 valid bit, and 1 dirty (owner) bit

- Read from main memory by processor i:
  - If dirty-bit OFF then { read from main memory; turn p[i] ON; }
  - if dirty-bit ON then { recall line from dirty proc (cache state to shared); update memory; turn dirty-bit OFF; turn p[i] ON; supply recalled data to i;}
- Write to main memory by processor i:
  - If dirty-bit OFF then { supply data to i; send invalidations to all caches that have the block; turn dirty-bit ON; turn p[i] ON; ... }
  - ...

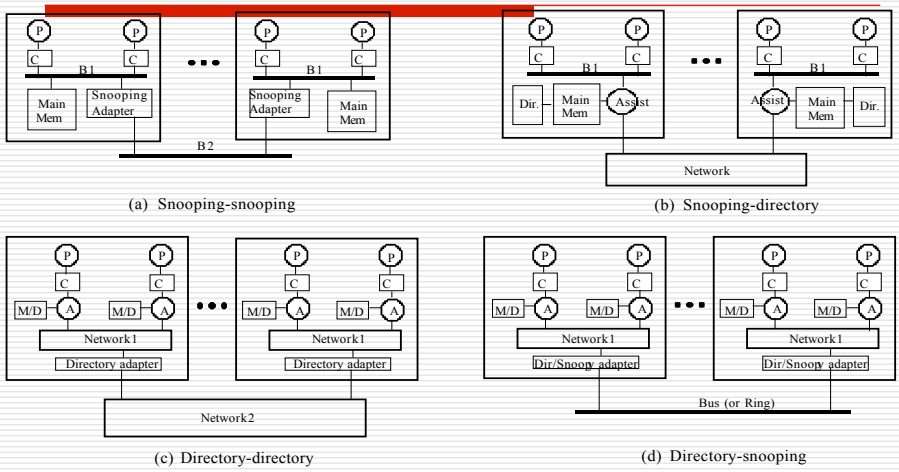
Basic Directory Transactions



A Popular Middle Ground

- Two-level "hierarchy"
- Individual nodes are multiprocessors, connected non-hiearchically
  - e.g. mesh of SMPs
- Coherence across nodes is directory-based
  - Directory keeps track of nodes, not individual processors
- Coherence within nodes is snooping or directory
  - Orthogonal, but needs a good interface of functionality
- Examples:
  - Convex Exemplar: directory-directory
  - Sequent, Data General, HAL: directory-snoopy
- SMP on a chip?

Example Two-level Hierarchies



(a) Snooping-snooping (b) Snooping-directory

(c) Directory-directory (d) Directory-snooping

Advantages of Multiprocessor Nodes

- Potential for cost and performance advantages
  - Amortization of node fixed costs over multiple processors
    - applies even if processors simply packaged together but not coherent
  - Can use commodity SMPs
  - Less nodes for directory to keep track of
  - Much communication may be contained within node (cheaper)
  - Nodes prefetch data for each other (fewer "remote" misses)
  - Combining of requests (like hierarchical, only two-level)
  - Can even share caches (overlapping of working sets)
  - Benefits depend on sharing pattern (and mapping)
    - Good for widely read-shared: e.g. tree data in Barnes-Hut
    - Good for nearest-neighbor, if properly mapped
    - Not so good for all-to-all communication

Disadvantages of Coherent MP Nodes

- Bandwidth shared among nodes
  - all-to-all example
  - Applies to coherent or not
- Bus increases latency to local memory
- With coherence, typically wait for local snoop results before sending remote requests
- Snoopy bus at remote node increases delays there too, increasing latency and reducing bandwidth
- May hurt performance if sharing patterns don't comply

Outline

- Today:
  - Overview of directory-based approaches
  - Inherent program characteristics
  - Correctness, including serialization and consistency
- Next: Implementation case study
  - NUMAchine

Scaling Issues

- Memory and directory bandwidth
  - Centralized directory is bandwidth bottleneck, just like centralized memory
  - How to maintain directory information in distributed way?
- Performance characteristics
  - Traffic: no. of network transactions each time protocol is invoked
  - Latency = no. of network transactions in critical path
- Directory storage requirements
  - Number of presence bits grows as the number of processors
- How directory is organized affects all these, performance at a target scale, as well as coherence management

Insight into Directory Requirements

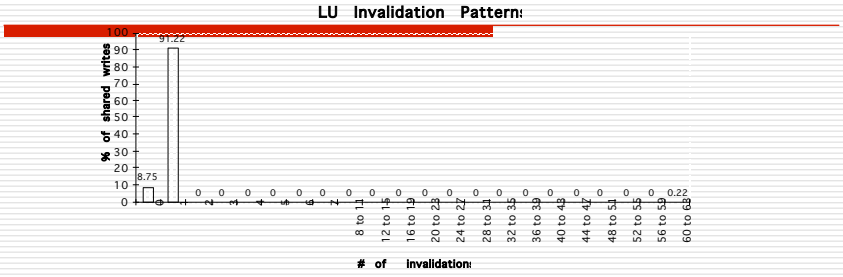
- If most misses involve O(P) transactions, might as well broadcast!

=> Study Inherent program characteristics:

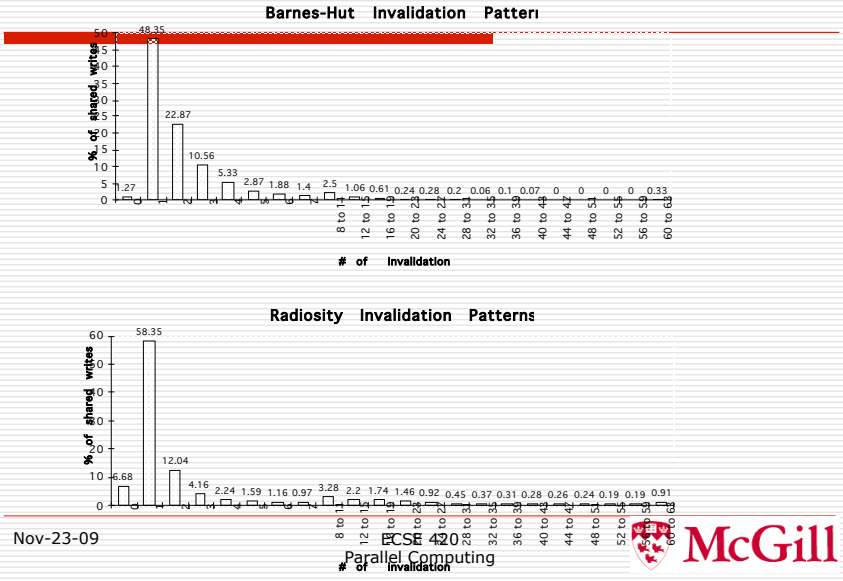
  - Frequency of write misses?
  - How many sharers on a write miss
  - How these scale

- Also provides insight into how to organize and store directory information

Cache Invalidation Patterns



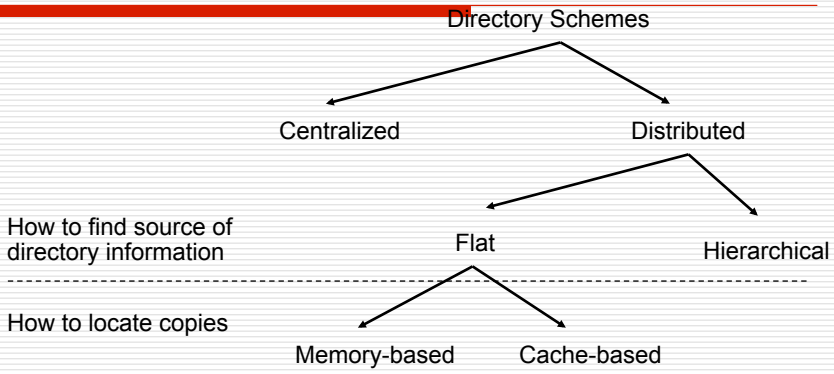
Cache Invalidation Patterns



Sharing Patterns Summary

- Generally, few sharers at a write, scales slowly with P
  - Code and read-only objects (e.g, scene data in Raytrace)
    - No problems as rarely written
  - Migratory objects (e.g., cost array cells in LocusRoute)
    - Even as # of PEs scale, only 1-2 invalidations
  - Mostly-read objects (e.g., root of tree in Barnes)
    - Invalidations are large but infrequent, so little impact on performance
  - Frequently read/written objects (e.g., task queues)
    - Invalidations usually remain small, though frequent
  - Synchronization objects
    - Low-contention locks result in small invalidations
    - High-contention locks need special support (SW trees, queueing locks)
- Implies directories very useful in containing traffic
  - If organized properly, traffic and latency shouldn't scale too badly
- Suggests techniques to reduce storage overhead

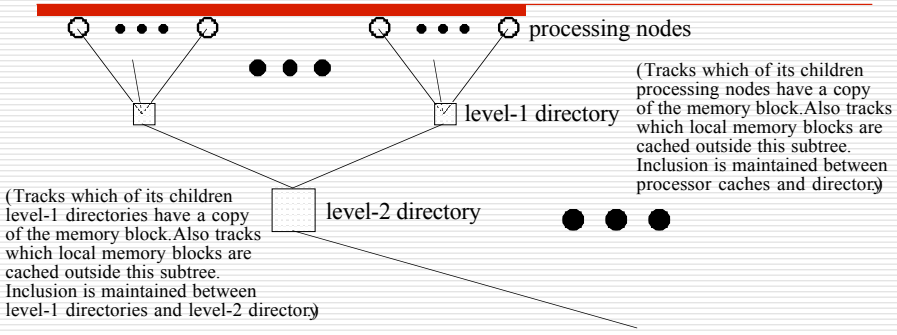
Organizing Directories



How to Find Directory Information

- Centralized memory and directory - easy: go to it
  - But not scalable
- Distributed memory and directory
  - Flat schemes
    - Directory distributed with memory: at the home
    - Location based on address (hashing): network xaction sent directly to home
  - Hierarchical schemes
    - ??

How Hierarchical Directories Work



- Directory is a hierarchical data structure
  - Leafs are processing nodes, internal nodes just directory
  - Logical hierarchy, not necessarily physical
    - (can be embedded in general network)

Find Directory Info (cont)

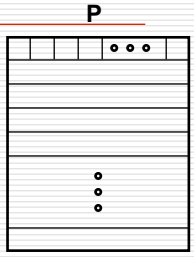
- Distributed memory and directory
  - Flat schemes
    - Hash
  - Hierarchical schemes
    - Node's directory entry for a block says whether each subtree caches the block
    - To find directory info, send "search" message up to parent
      - Routes itself through directory lookups
    - Like hiearchical snooping, but point-to-point messages between children and parents

How Is Location of Copies Stored?

- Hierarchical Schemes
  - Through the hierarchy
  - Each directory has presence bits child subtrees and dirty bit
- Flat Schemes
  - Vary a lot
  - Different storage overheads and performance characteristics
  - Memory-based schemes
    - Info about copies stored all at the home with the memory block
    - Dash, Alewife , SGI Origin, Flash
  - Cache-based schemes
    - Info about copies distributed among copies themselves
      - Each copy points to next
    - Scalable Coherent Interface (SCI: IEEE standard)

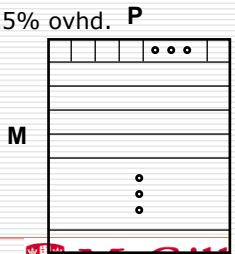
Flat, Memory-based Schemes

- Info about copies colocated with block at the home
- Performance Scaling
  - Traffic on a write: proportional to # sharers
  - Latency on write: can issue invalidations to sharers in parallel
- Storage overhead
  - Simplest representation: *full bit vector*, i.e. one presence bit per node
  - Storage overhead doesn't scale well with P; 64-byte line implies
    - 64 nodes: 12.7% ovhd.
    - 256 nodes: 50% ovhd.; 1024 nodes: 200% ovhd.
  - For M memory blocks in memory, storage overhead is proportional to P*M



Reducing Storage Overhead

- Optimizations for full bit vector schemes
  - Increase cache block size (reduces storage overhead proportionally)
  - Use multiprocessor nodes (bit per mp node, not per processor)
  - Still scales as P*M, but reasonable for all but very large machines
    - 256-procs, 4 per cluster, 128B line: 6.25% ovhd.
- Reducing "width"
  - Addressing the P term?
- Reducing "height"
  - Addressing the M term?

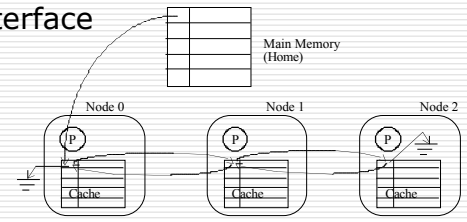


Storage Reductions

- Width observation:
  - Most blocks cached by only few nodes
  - Don't have a bit per node, but entry contains a few pointers to sharing nodes
  - P=1024 => 10 bit ptrs, can use 100 pointers and still save space
  - Sharing patterns indicate a few pointers should suffice (five or so)
  - Need an overflow strategy when there are more sharers
- Height observation:
  - Number of memory blocks >> number of cache blocks
  - Most directory entries are useless at any given time
  - Organize directory as a cache, rather than having one entry per memory block

Flat, Cache-based Schemes

- How they work:
  - Home only holds pointer to rest of directory info
  - Distributed linked list of copies, weaves through caches
    - Cache tag has pointer, points to next cache with a copy
  - On read, add yourself to head of the list (comm. needed)
  - On write, propagate chain of invals down the list
- Scalable Coherent Interface
  - Doubly linked list
  - IEEE Standard



Scaling Properties (Cache-based)

- Traffic on write: proportional to number of sharers
- Latency on write: proportional to number of sharers!
  - Don't know identity of next sharer until reach current one
  - Also assist processing at each node along the way
  - (even reads involve more than one other assist: home and first sharer on list)
- Storage overhead: quite good scaling along both axes
  - Only one head ptr per memory block
    - Rest is all prop to cache size
- Very complex!!!

Summary of Directory Schemes

- Flat Schemes:
- Issue (a): finding source of directory data
  - Go to home, based on address
- Issue (b): finding out where the copies are
  - Memory-based: all info is in directory at home
  - Cache-based: home has pointer to first element of distributed linked list
- Issue (c): communicating with those copies
  - Memory-based: point-to-point messages (perhaps coarser on overflow)
    - Can be multicast or overlapped
  - Cache-based: part of point-to-point linked list traversal to find them
    - Serialized
- Hierarchical Schemes:
  - All three issues through sending messages up and down tree
  - No single explict list of sharers
  - Only direct communication is between parents and children

Summary of Directory Schemes

- Directories offer scalable coherence on general networks
  - No need for broadcast media
- Many possibilities for organizing directory and managing protocols
- Hierarchical directories not used much
  - High latency, many network transactions, and bandwidth bottleneck at root
- Both memory-based and cache-based flat schemes are alive
  - For memory-based, full bit vector suffices for moderate scale
    - Measured in nodes visible to directory protocol, not processors

Issues for Directory Protocols

- Correctness
- Performance
- Complexity and dealing with errors

Discuss major correctness and performance issues that a protocol must address

Then delve into memory- and cache-based protocols, tradeoffs in how they might address (case studies)

Complexity will become apparent through this

Correctness

- Ensure basics of coherence at state transition level
  - Relevant lines are updated/invalidated/fetched
  - Correct state transitions and actions happen
- Ensure ordering and serialization constraints are met
  - For coherence (single location)
  - For consistency (multiple locations): assume sequential consistency
- Avoid deadlock, livelock, starvation
- Problems:
  - Multiple copies AND multiple paths through network (distributed pathways)
  - Unlike bus and non cache-coherent (each had only one)
  - Large latency makes optimizations attractive
    - Increase concurrency, complicate correctness

Coherence: Serialization to a Location

- Need entity that sees op's from many procs
- Bus:
  - Multiple copies, but serialization by bus imposed order
- Scalable MP without coherence:
  - Main memory module determined order
- Scalable MP with cache coherence
  - Home memory good candidate
    - All relevant ops go home first
  - But multiple copies
    - Valid copy of data may not be in main memory
    - Reaching main memory in one order does not mean will reach valid copy in that order
    - Serialized in one place doesn't mean serialized wrt all copies

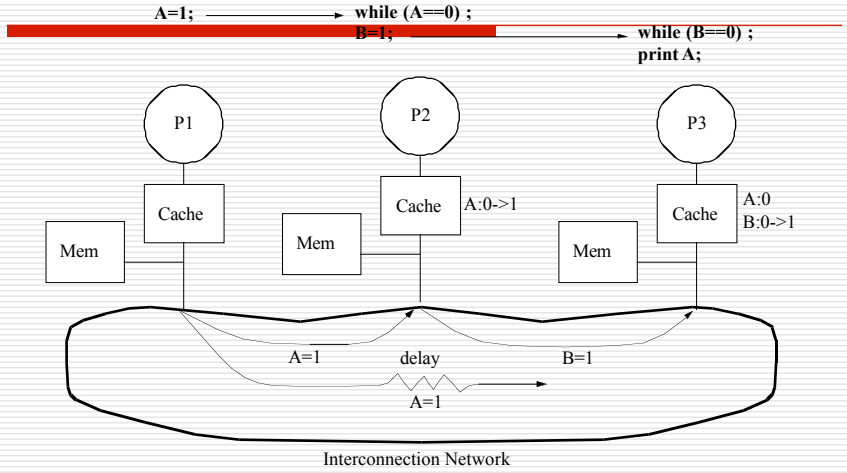
Basic Serialization Solution

- Use additional 'busy' or 'pending' directory states
- Indicate that operation is in progress, further operations on location must be delayed
  - buffer at home
  - buffer at requestor
  - NACK and retry
  - forward to dirty node

Sequential Consistency

- Bus-based:
  - Write completion: wait till gets on bus
  - Write atomicity: bus plus buffer ordering provides
- Non-coherent scalable case
  - Write completion: needed to wait for explicit ack from memory
  - Write atomicity: easy due to single copy
- With multiple copies and distributed network paths
  - Write completion: need explicit acks from copies themselves
  - Writes are not easily atomic
  - ... in addition to issues with bus-based and non-coherent

Write Atomicity Problem



Basic Solution

- In invalidation-based scheme, block owner (mem to $) provides appearance of atomicity by waiting for all invalidations to be ack'd before allowing access to new value.
- Much harder in update schemes!

Deadlock, Livelock, Starvation

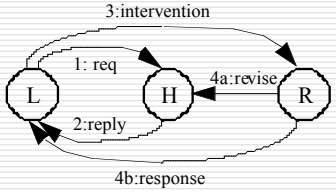
- Request-response protocol
- Similar issues to those discussed earlier
  - A node may receive too many messages
  - Flow control can cause deadlock
  - Separate request and reply networks with request-reply protocol
  - Or NACKs, but potential livelock and traffic problems
- New problem: protocols often are not strict request-reply
  - e.g. rd-excl generates inval requests (which generate ack replies)
  - Other cases to reduce latency and allow concurrency
- Must address livelock and starvation too
- Will see how protocols address these correctness issues

Performance

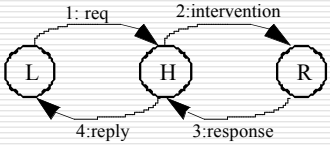
- Latency
  - Protocol optimizations to reduce network transactions in critical path
  - Overlap activities or make them faster
- Throughput
  - Reduce number of protocol operations per invocation
- Care about how these scale with the number of nodes

Protocol Enhancements for Latency

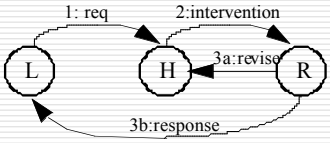
- Forwarding messages: memory-based protocols



(a) Strict request-reply



(a) Intervention forwarding



(a) Reply forwarding

Intervention is like a req, but issued in reaction to req. and sent to cache, rather than memory.

Other Latency Optimizations

- Throw hardware at critical path
  - SRAM for directory (sparse or cache)
  - Bit per block in SRAM to tell if protocol should be invoked
- Overlap activities in critical path
  - Multiple invalidations at a time in memory-based
  - Overlap invalidations and acks in cache-based
  - Lookups of directory and memory, or lookup with transaction
    - Speculative protocol operations

Increasing Throughput

- Reduce the number of transactions per operation
  - invals, acks, replacement hints
  - All incur bandwidth and assist occupancy
- Reduce assist occupancy or overhead of protocol processing
  - Transactions small and frequent, so occupancy very important
- Pipeline the assist (protocol processing)
- Many ways to reduce latency also increase throughput
  - e.g. forwarding to dirty node, throwing hardware at critical path...

Complexity

- Cache coherence protocols are complex
- Choice of approach
  - Conceptual and protocol design versus implementation
- Tradeoffs within an approach
  - Performance enhancements often add complexity, complicate correctness
    - More concurrency, potential race conditions
    - Not strict request-reply
- Many subtle corner cases
  - BUT, increasing understanding/adoption makes job easier
  - Automatic verification is important but hard

Summary

- In directory protocol there is substantial implementation complexity below the logical state diagram
  - Directory vs cache states
  - Transient states
  - Race conditions
  - Conditional actions
  - Speculation
- Real systems reflect interplay of design issues at several levels