Cache Coherence in Scalable Machines

CS 258, Spring 99
David E. Culler
Computer Science Division
U.C. Berkeley

4/2/99

Context for Scalable Cache Coherence

Scalable Networks
- many simultaneous transactions
- Caches naturally replicate data
- coherence through bus snooping protocols - consistency

Need cache coherence protocols that scale!
- no broadcast or single point of order

Generic Solution: Directories

• Maintain state vector explicitly
  - associate with memory block
  - 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

A 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 (invalidation/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
  - on 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

One Approach: 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 the 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

Basic Directory Transactions

- 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

A Popular Middle Ground

- Two-level “hierarchy”
- Individual nodes are multiprocessors, connected non-hierarchically
  - 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

- 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

Advantages of Multiprocessor Nodes
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
- Wed 4/7 Greg Papadopoulos
- Fri 4/9 Implementation
  - case Studies: SGI Origin2000, Sequent NUMA-Q
  - discuss alternative approaches in the process
- Later
  - Synchronization
  - Implications for parallel software
  - Relaxed memory consistency models
  - Alternative approaches for a coherent shared address space

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 issues

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

- Barnes-Hut Invalidation Patterns
- Ocean Invalidation Patterns
- Radiosity 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, queuing
      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 source of directory information
- Centralized
- Distributed

How to locate copies
- Flat
- Hierarchical

Directory Schemes

Centralized
Distributed

How Hierarchical Directories Work

Processing nodes

• Directory is a hierarchical data structure
  - leaves 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
    - directory distributed with memory: at the home
    - location based on address (hashing); network action
      sent directly to home
  - hierarchical schemes
    - ??

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 (SCl, IEEE standard)
Flat, Memory-based Schemes

- Info about copies colocated with block at the home
  - Just like centralized scheme, except distributed
- Performance Scaling
  - Traffic on a write: proportional to number of 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
- 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 invalu down the list
- Scalable Coherent Interface (SCI) IEEE Standard
  - Doubly linked list

Summary of Directory Organizations

- 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 explicit list of sharers
  - Only direct communication between parents and children

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!!!

Reducing 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 Approaches

- 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
  - will examine case studies of each

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

Correctness: Serialization to a Location

- Need entity that sees op’s from many procs
- bus:
  - write completion: wait till gets on bus
  - write atomicity: bus plus buffer ordering provides
- 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
- now, with multiple copies and distributed network pathways
  - write completion: need explicit acks from copies themselves
  - writes are not easily atomic
  - ... in addition to earlier issues with bus-based and non-coherent
Write Atomicity Problem

Interconnection Network

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 xactions 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

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 much easier
  - automatic verification is important but hard
- Let’s look at memory- and cache-based more deeply through case studies