Recall: Sequential Consistency Example

<table>
<thead>
<tr>
<th>Processor 1</th>
<th>Processor 2</th>
<th>One Consistent Serial Order</th>
</tr>
</thead>
<tbody>
<tr>
<td>LD₁ A ⇒ 5</td>
<td>LD₅ B ⇒ 2</td>
<td>LD₁ A ⇒ 5</td>
</tr>
<tr>
<td>LD₂ B ⇒ 7</td>
<td>...</td>
<td>LD₂ B ⇒ 7</td>
</tr>
<tr>
<td>ST₁ A,6</td>
<td>LD₆ A ⇒ 6</td>
<td>ST₁ A,6</td>
</tr>
<tr>
<td>...</td>
<td>ST₄ B,21</td>
<td>LD₅ B ⇒ 2</td>
</tr>
<tr>
<td>LD₃ A ⇒ 6</td>
<td>...</td>
<td>LD₆ A ⇒ 6</td>
</tr>
<tr>
<td>LD₄ B ⇒ 21</td>
<td>LD₇ A ⇒ 6</td>
<td>ST₄ B,21</td>
</tr>
<tr>
<td>ST₂ B,13</td>
<td>...</td>
<td>LD₃ A ⇒ 6</td>
</tr>
<tr>
<td>ST₃ B,4</td>
<td>LD₈ B ⇒ 4</td>
<td>ST₂ B,13</td>
</tr>
</tbody>
</table>

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.

Implications for Implementation

- Consider Distributed Protocol:
  - Node X: WREQ A
  - INV A
  - ACK A
  - Node Y: WACK A
  - ACK A
  - Inv A
  - Node Z: INV A

- Serious restrictions on when you can issue requests
  - Processor held up on store until:
    - Every processor cache invalidated!

- How to prove that this is sequentially consistent??
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 imposed by bus order
  - Timestamp snooping: serialization imposed by virtual time
• scalable MP without coherence:
  - main memory module determined order
• scalable MP with cache coherence
  - home memory good candidate
    » all relevant ops go home first
    » 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

Serialization: Filter Through Home Node?

• Need a serializing agent
  - home memory is a good candidate, since all misses go there first
• Having single entity determine order is not enough
  - it may not know when all actions for that operation are done everywhere

• Problem:
  - Home deals with write access before prev. is fully done
  - P1 should not allow new access to line until old one “done”

Basic Serialization Solution

• Use additional ‘busy’ or ‘pending’ directory and cache states
• Cache-Side: “Transaction Buffer”
  - Similar to memory load/store buffer in uniprocessor
  - Handles misordering in network:
    » E.g. Sent request, got invalidate first: wait to invalidate until get response from memory (either data or Nack)
    » Make sure that memory state machine has actual understanding of state of caches + network
• Memory-Side: Indicate that operation is in progress, further operations on location must be delayed or queued
  - buffer at home
  - buffer at requestor
  - Nack and retry
  - forward to dirty node
Recall: Ordering: Scheurich and Dubois

- Sufficient Conditions
  - every process issues mem operations in program order
  - after a write operation is issued, the issuing process waits for the write to complete before issuing next memory operation
  - after a read is issued, the issuing process waits for the read to complete and for the write whose value is being returned to complete (globally) before issuing its next operation
- How to get exclusion zone for directory protocol?
  - Clearly need to make sure that invalidations really invalidate copies
    - Keep state to handle reordering at client (previous slide’s problem)
  - While acknowledgements outstanding, cannot handle read requests
    - NAK read requests
    - Queue read requests

Achieving write exclusion zone

- Example for invalidation-based scheme:
  - block owner (home node) provides appearance of atomicity by waiting for all invalidations to be ack’d before allowing access to new value
  - As a result, write commit point becomes point at which WData leaves home node (after last ack received)
- Much harder in update schemes!

Livellock: Cache Side

- Window of Vulnerability: Time between return of value to cache and consumption by cache
  - Can lead to cache-side thrashing during context switching
  - Various solutions possible include stalling processor until result can be consumed by processor
  - Discussion of “Closing the Window of Vulnerability in Multiphase Memory Transactions” by Kubiatowicz, Chaiken, and Agarwal
- Solution:
  - Provide Transaction buffers to track state of outstanding requests
  - Stall context switching and invalidations selectively when thrashing detected (presumably rare)

Livellock: Memory Side

- What happens if popular item is written frequently?
  - Possible that some disadvantaged node never makes progress!
  - This is a memory-side thrashing problem
- Examples:
  - High-conflict lock bit
  - Scheduling queue
- Solutions?
  - Ignore
    - Good solution for low-conflict locks
    - Bad solution for high-conflict locks
  - Software queueing: Reduce conflict by building a queue of locks
    - Example: MCS Lock (“Mellor-Crummey-Scott”)
  - Hardware queueing at directory: Possible scalability problems
    - Example: QOLB protocol (“Queue on Lock Bit”)
    - Natural fit to SCI protocol
  - Escalating priorities of requests (SGI Origin)
    - Pending queue of length 1
    - Keep item of highest priority in that queue
    - New requests start at priority 0
    - When NACK happens, increase priority
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
  - Reduce the residency time in the directory controller
    » Faster hardware, etc
• 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
• Relaxing Consistency, e.g. Stanford Dash:
  - Write request when outstanding reads: home node gives data to requestor, directs invalidation acks to be returned to requester
  - Allow write to continue immediately upon receipt of data
  - Does not provide Sequential consistency (provides release consistency), but still write atomicity:
    » Cache can refuse to satisfy intervention (delay or NACK) until all acks received
    » Thus, writes have well defined ordering (coherence), but execution need not necessarily sequentially consistent (instructions after write not delayed until write completion)

Increasing Throughput

• Reduce the number of transactions per operation
  - inval, acks, replacement hints
  - all incur bandwidth and assist occupancy
• Reduce “occupancy” (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…
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 invalidation requests (which generate ack replies)
  - Other cases to reduce latency and allow concurrency

Mechanisms for reducing depth

Example of two-network protocols:
Only Request-Response (2-level responses)

Deadlock Issues with Protocols

- Consider dual graph of message dependencies
  - Nodes: Networks, Arcs: Protocol actions
  - Number of networks = length of longest dependency
  - Must always make sure response (end) can be absorbed!
**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

**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**

![Diagram of two-level hierarchies](image)

**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

Summary

- Issues for Distributed Directories
  - Correctness
  - Performance
  - Complexity and dealing with errors
- Serialization Solutions
  - Use additional 'busy' or 'pending' directory and cache states
  - Cache-Side: "Transaction Buffer"
  - Memory-Side: Indicate that operation is in progress, further operations on location must be delayed or queued
- Performance Enhancements
  - Reduce number of hops
  - Reduce occupancy of transactions in memory controller
- Deadlock Issues with Protocols
  - Many protocols are not simply request-response
  - Consider Dual graph of message dependencies
    - Nodes: Networks, Arcs: Protocol actions
    - Consider maximum depth of graph to discover number of networks