Formal Verification and its Impact on the Snooping versus Directory Protocol Debate

Milo M. K. Martin
University of Pennsylvania
milom@cis.upenn.edu

Acknowledgements

• Many thanks to my collaborators
  • Mark Hill, David Wood, Mike Marty @ Wisconsin
  • Dan Sorin @ Duke
  • Alan Hu and Jesse Bingham @ UBC
  • Rajeev Alur, Sebastian Burckhardt @ Penn

• Supported by
  • IBM Graduate Fellowship, Sun, Intel
  • NSF

Overview

• Multiprocessor cache coherence protocols
  • Allows a multiprocessor look like a multi-programmed uniprocessor to software
  • Complex, concurrent, and performance critical
  • No consensus on general design approach
    • Multi-decade debate still raging
• Formal verification
  • Used in finding bugs in cache coherence protocols
  • A great success in real-world use of formal verification
• This presentation:
  • Revisiting debate in the context of formal verification
  • Some observations on protocol design & verification

Caveats

• I’m not a verification expert
  • Primary expertise is computer architecture
  • Especially multiprocessor memory systems
  • Some dabbling in formal verification

• I’m only an academic
  • Limited industrial experience
  • But lots of conversations with designers

• Some of what I will say is controversial
  • Not all of it is new, as well

Outline

• Multiprocessors and coherence background
• Formal verification and coherence protocols
• Revisit the snooping vs directory protocol debate
• A new alternative: Token Coherence
• Conclusion

Multiprocessors

• Multiprocessors are becoming ubiquitous
  • All servers, multi-core desktops, multi-core embedded
  • After decades of research and niche deployment
• Why now?
  • Today’s workload (server and media workloads)
    • SQL and OpenGL most used “parallel languages”
  • Commodity multiprocessor software (e.g., Linux)
  • Power-efficient way to multiply performance
    • E.g., StrongARM 1Ghz → 200Mhz, 30x less power
    • Use 5 cores, 6x power reduction, same net speed
  • Difficult software transition from one to two cores
    • Much easier after that… exciting times
Multiprocessor Hardware

- Provide a shared-memory abstraction
  - Familiar and efficient for programmers

![Memory System](image-url)

Invalidation-based Cache-Coherence

- **Goal:** provide a “consistent” view of memory
- Permissions in each cache per block
  - One read/write -or- “exclusive block”
  - Many readers “shared block”
- Cache coherence protocols
  - Distributed & complex
  - Correctness critical
  - Performance critical
- **Races:** the main source of complexity
  - Requests for the same block at the same time

Snooping Protocols

- **Original designs**
  - Bus-based broadcast
  - High-speed point-to-point links
  - No (multi-drop) busses
  - Build “virtual bus”
  - Increasingly not globally synchronous
- **Other enhancements**
  - Split transaction
  - Multiple request and response interconnects
  - Snoop response combining
  - Distribute memory on each processor node

Two classes of multiprocessors

- **Snooping multiprocessors**
  - Uses broadcast
  - “Virtual bus” interconnect
  - Directly locate data (2 hops)
- **Directory-based multiprocessors**
  - Directory tracks writer or readers
  - Avoids broadcast
  - Avoids “virtual bus” interconnect
  - Indirection for cache-to-cache (3 hops)
- Method for ordering racing requests is key

Snooping Example

- **Requestor**
  - **Requestor**
  - **Read/Write**
  - **Home**

- **Virtual bus** (totally-ordered) Interconnect

Root

P₀

P₁

P₂

M₀
Snooping Example

Virtual bus (totally-ordered) Interconnect

Root ordered interconnect orders requests

Directory Protocols
- Send all requests to directory
- Avoids broadcast
  - "Scalable", but who cares?
  - Most systems sold are modest in size
  - Does not require interconnect ordering
- (Bad) alternative names:
  - "CC-NUMA"
  - "Distributed shared memory"
  - "Scalable cache coherence"
  - Why bad names? don’t capture the fundamental differences

Directory Example

Request

Fwd

Data

Done

No ordered interconnect, directory orders requests
The Debate: Snooping v. Directories
Which approach is “better”?
• Debated for 20+ years
• Mostly debated in terms of
  • Scalable performance
  • Performance
• Let’s revisit the debate in terms of
  • Design complexity
  • Verification’s impact on the above

Outline
• Multiprocessors and coherence background
  • Formal verification and coherence protocols
  • Revisit the snooping vs directory protocol debate
  • A new alternative: Token Coherence
• Conclusion

Formal Verification & Coherence Protocols
• Model the protocol at a high level
  • Abstract away some implementation details
  • Capture concurrent races
  • Find protocol bugs (earlier the better)
  • Alternative: verify implementation vs high-level model
• Multitude of formal techniques
  • Model checking, theorem proving, SAT solvers, etc.
• Apply to scaled down system
  • Few processors, two data values, two addresses, limited traces, etc.

Explicit Role of Formal Verification
• Post-design verification
  • Used more like traditional design verification
  • Can help find bugs, but many “false bugs”
  • Out of date or incomplete specification
  • Or previously found and fixed
  • Many case studies, e.g., [Hu et al., ICCD 1997]
• During-design verification
  • Model creation part of design specification process
  • “Formal verifiers” part of cross-functional design team
  • Find bugs early → easier, cleaner fixes
  • Becoming more common, fewer anecdotes

Implicit Role of Verification
• Once formal verification is part of design…
• Has implicit impact on the actual design
  • A series of bugs might change high-level design
  • Forces deep systematic think about the design
  • Gives designers confidence
  • Just making the model can find bugs (story)
• “Verifiability” becomes a design constraint
  • Designers react to it (story)
  • Encourages modular, cleaner, documented designs

Implicit Role of Verification (continued)
• Is a “verifiable” design a better design?
  • “principles of good design”, keeps designers honest
  • Avoid problems before “bugs” develop
  • Easier alternative? just trick the designers
• Design systems to be formally verified?
  • How might doing so affect low-level concurrent protocols?
  • What might such a coherence protocol look like?
  • I’ll talk about one possibility later in talk…
Two Desirable Coherence Properties

- What properties might a coherence protocol...
  - To make it “verifiable”
  - To make it simple
  - To make it flexible

- Two desirable decoupling properties
  - Decouple interconnect properties from protocol
  - Decouple consistency from coherence

Decouple Interconnect from Protocol (1 of 2)

- Unordered interconnections
  - Simple, modular interface
  - Deadlock avoidance via virtual networks
  - Constrains design and model the least

- Point-to-point ordered interconnects
  - Disallows adaptive routing
  - Reduces symmetry of model (small state space)
  - Not so bad, but better to avoid

- Most directory protocol fall into these categories

Decouple Interconnect from Protocol (2 of 2)

- Totally-ordered interconnects
  - Requires a bus or “virtual bus”, “snoop combining”
  - Sometimes timing sensitive
  - Complicate interface, implementation, modeling

- What protocols require this property?
  - Snooping (all)
  - Is “snooping” defined by broadcast or ordering?
  - Few directory protocols (e.g., GS320)

Decouple Coherence from Consistency

- Memory consistency models
  - Defines “consistent” view of memory
  - Coherence: for a single location
  - Consistency: ordering among multiple locations

- Example:
  
  Initial state: A = B = 0
  
  Thread #0
  while(A == 0) { /* nothing */ }
  Store B
  Load B
  
  Thread #1
  Store B ← 1
  Store A ← 1

- "Load B" should return?
  - Under sequential consistency, always one
  - Can return zero under weaker models

Enforcing A Memory Consistency Model

- Option #1
  - Coherence protocol provides “coherence invariant”
  - Single-reader/writer —or— multiple readers
  - Processor internally allows or disallows reorderings
  - All “sync” instructions internal to processor core
  - Example: Alpha 21364

- Option #2
  - Intertwine and disperse enforcement through system
  - Totally order all requests
  - Send “sync” instructions into memory system
  - Maybe write-through L1 caches in multi-core systems
  - Example: IBM Power4

Decoupling Implications

- For verification
  - Easier to model each piece independently & together
  - Reuse models over time

- For design
  - More compartmentalized
  - Easier incremental improvement over time
  - Reuse of design components
Revisiting Snooping vs Directory Protocols

- **Snooping Protocols**
  - Simple snooping is seductively simple
  - "Atomic" with simple bus
  - More aggressive implementations are quite complex
  - Violate the two decoupling properties

- **Directory Protocols**
  - Have the decoupling properties
  - Complex, but in all the ways formal methods can help
  - Better "complexity scalability" over time

Why Aren’t Directory Protocols More Common?

- **Complexity disconnect**
  - No evolutionary path to directory protocols
  - Radical design departure
  - Designers are good at incrementally improving working approaches over time

- **Scalability trap**
  - Previous idea: scalability at all costs!
  - Should only be a means to an end, not an end goal
  - "Scalable cache coherence" is synonymous with directory protocols

- **Often used to bridge between snooping systems**
  - Reputation for high latency

My Opinion on the Coherence Debate?

- I now advocate against snooping protocols
  - But for different reasons than others
  - i.e., not performance scalability
  - Main reason: decoupling properties

- **A reversal of my previous opinion!**
  - Previously, I explored evolving snooping protocols
    - [ASPL00, HPCA 2002]
  - Now, tightly-coupled directory protocols attractive

- **AMD’s Operton protocol is interesting**
  - "Directory-less" directory protocol
  - Glueless, point-to-point interconnect, non-scalable

- **Or, a new alternative…**

A New Alternative: Token Coherence [ISCA 2003]

- **A protocol design to be verified formally**
  - Fast, simple, flexible, too.

- **Decoupling correctness and performance**
  - Correctness substrate
    - Safety via token counting
    - Forward progress via persistent requests
  - Separate performance policies
  - Target the common case

- **Separate correctness and performance**
  - Example of “Better Then Worst-Case Design”

Key Observation: Token Counting

- **Explicitly encode permissions with tokens**
  - At all times, **all blocks have T tokens**
    - E.g., one token per processor
  - Components exchange tokens & data

- **Tokens: in caches, memory, or in transit**
  - Controls reading & writing of data
    - One or more to read
    - All tokens to write

  **Provides safety in all cases**
Token Counting Example

- Each memory block initialized with T tokens
- At least one token to read a block
- All tokens to write a block

Guaranteeing Starvation-Freedom

- Handle pathological cases
  - Infrequently invoked
  - Can be slow, inefficient, and simple
- When normal requests fail to succeed (4x)
  - Longer timeout and issue a persistent request
  - Request persists until satisfied
  - Table at each processor
  - “Deactivate” upon completion
- Implementation
  - Arbiter at memory orders persistent requests

Performance Policies

- Opportunities
  - Aggressively target the common case
  - Requests are just “hints” to move data & tokens
- Robust
  - Can’t cause “correctness” violations
  - A null or random policy is correct
  - Rely on correctness substrate
- Examples
  - TokenB - broadcast policy
  - TokenD - performance characteristics of directory
  - TokenM - predictive multicast protocols
  - TokenCMP [HPCA 2005] - multi-level coherence
    - “Flat for correctness, hierarchical for performance”

Ramifications of T.C. on Design Verification

- Divide and conquer complexity
  - Formally verified Token Coherence [HPCA 2005]
  - Difficult to quantify, but promising
  - All races handled uniformly (reissuing)
    - E.g. simple replacements (no handshake)
- Local invariants
  - Safety is response-centric; independent of requests
  - Locally enforced with tokens
- Further innovation → no correctness worries

Token Coherence vs Directory Protocols

- Similarities
  - Decouple interconnect from protocol
  - Decouple coherence from consistency
  - Token Coherence more explicitly gives you a “serial” coherence
- Differences
  - Token Coherence can avoid directory indirection
  - Token Coherence is more flexible, decoupled
  - However, Token Coherence has separate persistent requests, which add complexity

Result: an interesting alternative

Outline

- Multiprocessors and coherence background
- Formal verification and coherence protocols
- Revisit the snooping vs directory protocol debate
- A new alternative: Token Coherence
- Conclusion
Conclusions

- The age of multiprocessors and multi-core chips
  - Coherence protocol is key design to such designs
- Formal verification has an important role to play
  - Leverage formal methods early in design process
    - Both explicit and implicit benefits
- Two decoupling properties
  - Decouple interconnect from protocol
  - Decouple coherence and consistency
- Snooping vs directory protocols?
  - Directory protocols have these decoupling properties
  - Token Coherence further embraces them