Specification and Validation of Cache Coherence protocols

Huy-Nam Nguyen
Huy-Nam.Nguyen@bull.net

Bull S.A.S.
Outline

- Motivations
- Cache Coherence Protocols
- Protocol Modeling and Validation
- Formal Verifications of CC Protocols
- Dynamic Verification of CC Protocols
- Conclusions
Motivations

- Large-scale cache-coherent shared-memory systems
  - Distributed-memory nodes + private/shared caches
  - Scalable cache-coherence
  - Memory consistency models

- Issues in protocol design/verification
  - Level of abstraction => micro-architecture
  - Definition focused on “normal” sequences of events
  - Large number of transient states & combination
  - Handling of large and parameterized configurations
  - Unpredictability of events order
  - Probability of a specific sequence occurrence is in inverse exponential proportion to the sequence length
Cache Coherence Protocol

Diagram showing a network of processors (Proc) connected to levels 1 (L1) of memory and directories (Dir Memory) through interfaces (Interface). The network connects these elements, indicating a distributed cache coherence protocol.
Cache Coherence Protocol (cont’d)

- **Write-through vs Write-back strategies**
  - Performance dependence
    - Application (#writes)
    - Cache block granularity
  - High bandwidth requirements vs Complex CC-Protocols

- **Directory-based vs Source Snooping Coherence**
  - Scalability vs Simplicity
  - Directory-based protocol: Efficiency for Distributed Memory and Weak Consistency
  - *Directory Implementations*: Full/Limited-map presence vector vs (Centralized) Chain vs Virtual Tree Directories

- **Memory Consistency Models**
  - **Strict consistency**: \( R(x) = \text{Most recent } W(x) \)
  - **Sequential consistency**: Global sequential order compatible with local/process ordering
  - **Weak consistency**: Sequentially consistent access to synchronization variables
Protocol Modeling and Checking

- Abstract FSM model of protocol
  - Single cache line with M[O]ESI states
  - Transition modeling
    » Atomic transaction
    » Non-determinism to model open design decisions
  - Fix number of processors

- Protocol Specification
  - Type of Modeling
    » Asynchronous modeling (Murphi, Promela)
    » Synchronous modeling (Lotos)
  - Specification semantics / Languages
    » Process Algebra CCS (Milner) : Lotos
    » Communicating Sequential process (CSP) : Pomela
    » Set of FSMs specified by rules: Murphi
(Simple) FSM Modeling

Write-back/ownership MESI Model
Protocol Modeling and Checking

- **Communication Models**
  - Message classes for requests/responses
  - Assignment to Virtual Network/Channels

- **Properties checking**
  - Properties related to the semantics of the cache states (M[O]ESI)
  - **Safety properties**
    » Local safety conditions
    » Freedom from deadlock/livelock
  - **Liveness properties** : Accessibility to all states
Protocol Modeling and Checking

- Specification for Synthesis/Validation

- P-Table
  - Consistency Checking
  - Formal Specification
    - VPC
    - Formal Verification
  - Synthesis
    - RTL
    - Simulation
Formal Verification Methods

- Fix-point calculations
  - Composition cache FSMs
  - Symbolic traversal techniques
  - CTL model checking, e.g.
    » AG(EF(readable) & EF(writable))
    » AG(readable & data => AG(readable => data))
  - Extension of the limit of processors by induction
- Examples of application
  » Verification of the SGI Origin 2000 (4 cluster) (E. McMillan)
  » Verification of the Sun S3.mp system (Pong et als)
  » Other protocols (Dubois et als.)

- Explicit state enumeration (Dill)
  - Explicit traversal of state space with verification of invariant
  - Direct application (e.g. Murphi) or integration in a SVM programming environment (e.g. Teapot)
Formal Verification Methods (cont ’d)

- Infinite-state symbolic model checking
  - Abstract protocol described by EFSMs
    » Finite automaton with data variables ranging over integers
    » EFSM-transitions are linear transformations with global/local conditions as guards: \( G(x) \Rightarrow T(x,x') \)
    » Verification as accessibility (P.A. Abdulla/G. Delzanno)
      • Constraint representing unsafe states (e.g. \( E+M > 1 \))
      • Backward reasoning to deduce new constraints ... covering init state
    » Example of rule: \( E > 0 \Rightarrow E = E-1, M = M+1 \)
  - Strong similarity with the Algebraic method (D. Mentré)
    » Transformation of multiset by production rules
    » Verification of invariants, e.g. \( \text{Card}(W, p, *) \leq 1 \)

- Proof of Model consistency
  - Traditional simulation
  - Time stamping methods
Formal Verification Methods (cont’d)

- Graph-based verification of dead-lock
  - Extraction of resource dependence graph
  - Loop-Free Checking by Graph transversal
  - Non-dependence of terminal requests/responses
  - Sufficient conditions for deadlock-free
Dynamic Verification

- Necessary to complement formal verification for the validation of large configurations

Simulation Environment
  - Interconnection Fabric
  - Terminal (Cache/Memory) Transactors

Automatic Checking of Coherence/Consistency
  - [Biased] Random generation of Requests/Responses
  - On-the-fly checking of coherence
  - Consistency checking of memory by shadow memory

Usage of Simulation accelerated hardware
  - Increase number of cycles (Billion of cycles)
  - Favor state space exploration and coverage
  - Replay from snapshots for debugging
Conclusions

- **Benefits from earlier protocol validation**
  - Force accurate/detailed protocol specification
  - Verification in conjunction with micro-architecture
  - High cost in checking and fixing protocol errors at RTL

- **Issues in abstract protocol modeling/checking**
  - Limitation of the single cache line model
  - Mostly incomplete models
  - Limitation of system configuration
  - Complement dynamic validation is mandatory

- **On-Going applications in the TSAR project**
  - Application to an IN-CC protocol
  - Support both Validation and Protocol Engine Synthesis