Overview

- Motivation
- Background
- Parametric Verification
- Design Guidelines
- PV-MOESI vs OP-MOESI
- Results
- Conclusion
Issue with Coherence Protocols

Difficult to automatically verify for many core systems

Better performance ⇒ Complex protocols ⇒ Difficult to formally verify
GOAL

Architect arbitrarily large flat protocols such that they can be verified using a mostly-automated methodology
Overview

- Motivation
- **Background**
- Parametric Verification
- Design Guidelines
- PV-MOESI vs OP-MOESI
- Results
- Conclusion
Coherence Protocols

Two Primary Categories: Snooping & Directory-based

(ex. MSI, MESI, MOESI, MESIF)
Formally verifying a coherence protocol using State Space Exploration and Theorem Proving.
State Space Exploration
(with Murphi)
GOAL

Architect arbitrarily large flat protocols such that they can be verified using a mostly-automated methodology
GOAL

Architect arbitrarily large flat protocols such that they can be verified using a mostly-automated methodology
Model Protocol in Murphi
Check invariants
Model Protocol in Murphi

Check invariants
Murphi Processor Node State Definition

**Cache line State**

Input messages \[\rightarrow\] Cache line State \[\rightarrow\] Output transition/messages
MOESI protocol
Processor
Cache Controller
`MOESI` protocol
Processor
Cache Controller
MOESI protocol
Processor
Cache Controller
MOESI protocol
Processor
Cache Controller
MOESI protocol
Processor
Cache Controller
Model Protocol in Murphi

Check invariants
Check invariants

1. **Permission Invariant:** Single-Writer, Multiple-Reader
2. **Data Invariant:** Read returns value of last write
Check invariants

1. **Permission Invariant:** Single-Writer, Multiple-Reader
2. **Data Invariant:** Read returns value of last write
Cache Line

ST

SINGLE Writer
Multiple Reader
Check invariants

1. Permission Invariant: Single-Writer, Multiple-Reader
2. Data Invariant: Read returns value of last write
Overview

- Motivation
- Background
- **Parametric Verification**
- Design Guidelines
- PV-MOESI vs OP-MOESI
- Results
- Conclusion
PARAMETRIC VERIFICATION (PV)

Treat number of *nodes* as a *parameter* (using Abster tool)
Prove properties of design agnostic to *parameter size*

This process scales to many nodes and is almost fully automatic
Simple-PV Process Flow
How to design a readily verifiable Coherence Protocol
Create model w/ small # nodes

1. Make Param. Model (Abster)

   Fail

   NOT VERIFIABLE!

   Success

   2. Model Check (using Murphi)

      Success

      Verification Success

      Fail

      Counter Example

      Fix Bug

      Yes

      Bug in Protocol?

      No

      Refinable?

      Yes

      Fix Bug

      No

      State Space Explosion

      NOT VERIFIABLE!

      Refinable?

      Yes

      3. Refine Model (Manual)

      No
1. Make Param. Model (Abster)

- Success
- Fail

2. Model Check (using Murphi)

- Success
- Fail

Counter Example

- Yes: Fix Bug
- No: Bug in Protocol?

3. Refine Model (Manual)

- Yes: Refinable?
- No: State Space Explosion

NOT VERIFIABLE!
Automatically Create Parametric Model

- Create parametric model from non-parametric model
  - N concrete nodes -> 2 concrete nodes + “Other Node” (N-2)
  - Abster automatic abstraction tool
- Abster over-approximates the behavior of the N-2 nodes
- If Abster fails, modify protocol until it is compatible
N-2 parameterized nodes
1. Make Param. Model (Abster)

Create model w/ small # nodes

Suc
c

Success
Fix
Bug

Fail

Success

2. Model Check (using Murphi)

2. Model Check (using Murphi)

Success

Verification Success

Fail

NOT VERIFIABLE!

3. Refine Model (Manual)

3. Refine Model (Manual)

Fail

Counter Example

Fix
Bug

Yes

Bug in Protocol?

No

Refinable?

Refinable?

No

State Space Explosion

NOT VERIFIABLE!

NOT VERIFIABLE!
Automatically Model Check the model (MURPHI)
Create model w/ small # nodes

1. Make Param. Model (Abster)
   - Success
   - Fail

2. Model Check (using Murphi)
   - Success
   - Fail
     - Counter Example
     - Not verifiable!
     - State Space Explosion

3. Refine Model (Manual)
   - Fix Bug
     - Yes
     - Bug in Protocol?
       - Yes
       - Refinable?
         - Yes
         - Verification Success
         - No
       - No
     - No
   - No

Simple-PV Process Flow
Manually Refine the Model

• Over-approximation leads to spurious invariant violations

• Must modify behavior of “Other node”

• **KEY:** Add constraint and check their validity
  • Add invariant *(lemma)* – must be true for non-abstracted model
  • Check on the concrete nodes
System Architecture for PVCoherence
Memory

Inclusive L2 $

Inclusive Directory $

Network on Chip

L1$ Core 0

L1$ Core 1

L1$ Core 2
- LD Miss
- ST Miss
- Eviction

- $\rightarrow$ GetS/GetE
- $\rightarrow$ GetM
- $\rightarrow$ PutM
- $\rightarrow$ PutO
- $\rightarrow$ PutE
Network on Chip

VC0 Requests
VC1 Responses
VC2 Forwards

•
•
•
Objective

during protocol design
Overview

- Motivation
- Background
- Parametric Verification
- **Design Guidelines**
- PV-MOESI vs OP-MOESI
- Results
- Conclusion
Guidelines
for Simple-PV compliance
Guidelines
for Simple-PV compliance

#1: Identical Nodes
#2: No variables must depend on number of Nodes
#3: No ordering over list/queue sized by number of nodes
#4: Should not parameterize buffers/queues in more than 1-dim.
Identical Nodes

- Memory
- Inclusive L2
- Inclusive Directory
- Network on Chip
- LIS
- Core 0
- LIS
- Core 1
- LIS
- Core 2
- ...

Heterogeneous Nodes

Inclusive L2

Network on Chip

Core 0

Core 1

Core 2

L1

Inclusive Directory
Guidelines for Simple-PV compliance

#1: Identical Nodes

#2: No variables must depend on number of Nodes

#3: No ordering over list/queue sized by number of nodes

#4: Should not parameterize buffers/queues in more than 1-dim.
N-1 Nodes to track
(N not known)
**N-1 Nodes to track**
(N not known)

Cannot compare with **parameterized** value or carry out math operations
N-1 Nodes to track
(N not known)

Replace with sharer set
(bit vector)
Guidelines for Simple-PV compliance

#1: Identical Nodes

#2: No variables must depend on number of Nodes

#3: No ordering over list/queue sized by number of nodes

#4: Should not parameterize buffers/queues in more than 1-dim.
Inclusive L2 $\rightarrow$ Inclusive Directory $\rightarrow$ Network on Chip $\rightarrow$ Independent FIFO Request Queue

Independent FIFO Request Queue is permitted by this method.
Inclusive L2 $ 

Inclusive Directory $ 

Network on Chip 

Core 0 Core 1 Req2 Req1 Req0 

Shared Request Queue 

FIFO 

UNORDERED 

LIS Core 0 

LIS Core 1 

LIS Core 0 

LIS Core 0 

LIS Core 0
Guidelines for Simple-PV compliance

#1: Identical Nodes

#2: No variables must depend on number of Nodes

#3: No ordering over list/queue sized by number of nodes

#4: Should not parameterize buffers/queues in more than 1-dim.
NO Buffer(N)(N) Practical limitation
L2 (Directory) collects all invalidation Acks
L2 Sends aggregate Ack to Core 0

GetM

Ack + data
Overview

- Motivation
- Background
- Parametric Verification
- Design Guidelines
- **PV-MOESI vs OP-MOESI**
- Results
- Conclusion
Optimizations
(OP-MOESI)
<table>
<thead>
<tr>
<th>Optimization</th>
<th>Compatible with SimplePV?</th>
<th>Impact</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adding <strong>Exclusive</strong> State</td>
<td>Y</td>
<td>NO IMPACT</td>
</tr>
<tr>
<td>Adding <strong>Owned</strong> state</td>
<td>Y</td>
<td>Add 2 lemmas</td>
</tr>
<tr>
<td>Adding <strong>Self-Upgrade</strong></td>
<td>Y</td>
<td>Add lemma</td>
</tr>
<tr>
<td>Adding <strong>Silent Evictions</strong></td>
<td>Y</td>
<td>Add lemma</td>
</tr>
<tr>
<td>Removing the completion messages for <strong>GetS</strong> when data response comes from L2$</td>
<td>Y</td>
<td>Add lemma</td>
</tr>
<tr>
<td>Removing the completion messages for <strong>GetM</strong></td>
<td>N</td>
<td>--</td>
</tr>
</tbody>
</table>
OP-MOESI to PV-MOESI

• To ensure successful abstraction by Abster...
  • For GetM
    • Replace response counter with sharer set
    • Let L2 collect Acks and send aggregated Ack to requesting L1
  • Remove point-to-point ordering in all VCs and avoid races by adding extra messages or transient states but without blocking L1
OP-MOESI to PV-MOESI

• To ensure successful verification by Murphi...

  • Problem: multiple in-flight GetM requests without ordering

  • Solution: L2 blocks other subsequent requests whenever it receives a GetM request until it receives a Completion message from the requesting L1

  • Impact: performance decrease due to blocking at L2
Evaluation:

**OP-MOESI vs. PV-MOESI**
Runtime

< 1% overhead for all benchmarks
Network Traffic Overhead

Few increases in Network Traffic (< 20%)
Performance Scalability

Scalable in both directions (up & down)
Storage Overhead

Overhead is generally **negligible**
CONCLUSION

- Design of **parametrically verifiable** coherence protocols is possible given that the **guidelines** introduced here are adhered to

- There is no significant performance drawbacks or storage overheads

- **Automation** is key
Debate

- Is it necessary for a protocol to be parametrically verifiable? There are a few design corners that are cut to make such PV-compliant protocols work. Is this worth it?