CCICheck: Using μhb Graphs to Verify the Coherence-Consistency Interface

Yatin A. Manerkar, Daniel Lustig, Michael Pellauer*, and Margaret Martonosi

Princeton University *NVIDIA

MICRO-48
Coherence and Consistency

At a high level:

• **Coherence Protocols**: Propagation of writes to other cores

• **Consistency Models**: Ordering rules for visibility of reads and writes
Coherence and Consistency

Coherence Verifiers

Consistency Verifiers

Arch. Level
Coherence and Consistency

Coherence and consistency often interwoven

Arch. Level

μarch. Level
Coherence and Consistency

Coherence Verifiers
Ignore consistency even when protocol affects consistency!

Consistency Verifiers
Assume abstract coherence instead of protocol in use!

Coherence and consistency often interwoven

Arch. Level

μarch. Level
Coherence and consistency often interwoven

Coherence Verifiers
Ignore consistency even when protocol affects consistency!

Consistency Verifiers
Assume abstract coherence instead of protocol in use!

Coherence and consistency often interwoven

μarch. Level

Arch. Level
Motivating Example – “Peekaboo”
Motivating Example – “Peekaboo”

1. Invalidation before use
   - Repeated inv before use $\rightarrow$ livelock [Kubiatowicz et al. ASPLOS 1992]
Motivating Example – “Peekaboo”

1. Invalidation before use
   - Repeated inv before use → livelock [Kubiatowicz et al. ASPLOS 1992]

2. Livelock avoidance: allow destination core to perform one operation on data when it arrives, even if already invalidated [Sorin et al. Primer]
   - Does not break coherence
   - Sometimes intentionally returns stale data
Motivating Example – “Peekaboo”

1. Invalidation before use
   - Repeated inv before use $\rightarrow$ livelock [Kubiatowicz et al. ASPLOS 1992]

2. Livelock avoidance: allow destination core to perform **one** operation on data when it arrives, **even if already invalidated** [Sorin et al. Primer]
   - Does **not** break coherence
   - Sometimes **intentionally** returns stale data

3. Prefetching
Motivating Example – “Peekaboo”

1. Invalidation before use
   – Repeated inv before use $\rightarrow$ livelock [Kubiatowicz et al. ASPLOS 1992]

   **Individual Opt. $\rightarrow$ No violation**

   **Combination of Opts. $\rightarrow$ Violation!**

   – Does **not** break coherence

   – Sometimes **intentionally** returns stale data

3. Prefetching
Motivating Example – “Peekaboo”

• Consider mp with the livelock-avoidance mechanism:

<table>
<thead>
<tr>
<th>Core 0</th>
<th>Core 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>(i1) St x ← 1</td>
<td>(i3) Ld r1 ← [y]</td>
</tr>
<tr>
<td>(i2) St y ← 1</td>
<td>(i4) Ld r2 ← [x]</td>
</tr>
</tbody>
</table>

Under TSO: Forbid r1=1, r2=0

<table>
<thead>
<tr>
<th>Core 0</th>
<th>Core 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>x: Shared</td>
<td>x: Invalid</td>
</tr>
<tr>
<td>y: Modified</td>
<td>y: Invalid</td>
</tr>
<tr>
<td>[x] ← 1</td>
<td>r1 ← [y]</td>
</tr>
<tr>
<td>[y] ← 1</td>
<td>r2 ← [x]</td>
</tr>
</tbody>
</table>
Motivating Example – “Peekaboo”

• Consider `mp` with the livelock-avoidance mechanism:

<table>
<thead>
<tr>
<th>Core 0</th>
<th>Core 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>(i1) St [x] ← 1</td>
<td>(i3) Ld r1 ← [y]</td>
</tr>
<tr>
<td>(i2) St [y] ← 1</td>
<td>(i4) Ld r2 ← [x]</td>
</tr>
</tbody>
</table>

Under TSO: Forbid r1=1, r2=0

Core 0
- x: Shared
- y: Modified
- [x] ← 1
- [y] ← 1

Core 1
- x: Invalid
- y: Invalid
- r1 ← [y]
- r2 ← [x]
Motivating Example – “Peekaboo”

- Consider **mp** with the livelock-avoidance mechanism:

<table>
<thead>
<tr>
<th>Core 0</th>
<th>Core 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>(i1) St</td>
<td>(i3) Ld</td>
</tr>
<tr>
<td>x ← 1</td>
<td>r1 ← [y]</td>
</tr>
<tr>
<td>(i2) St</td>
<td>(i4) Ld</td>
</tr>
<tr>
<td>y ← 1</td>
<td>r2 ← [x]</td>
</tr>
</tbody>
</table>

Under TSO: Forbid r1=1, r2=0

Core 0
x: Shared
y: Modified

[x] ← 1
[y] ← 1

Prefetch x
Data (x = 0)

Core 1
x: Invalid
y: Invalid
r1 ← [y]
r2 ← [x]
Motivating Example – “Peekaboo”

- Consider \texttt{mp} with the livelock-avoidance mechanism:

<table>
<thead>
<tr>
<th>Core 0</th>
<th>Core 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{St} \texttt{[x]} $\leftarrow$ 1</td>
<td>\texttt{Ld} \texttt{r1} $\leftarrow$ \texttt{[y]}</td>
</tr>
<tr>
<td>\texttt{St} \texttt{[y]} $\leftarrow$ 1</td>
<td>\texttt{Ld} \texttt{r2} $\leftarrow$ \texttt{[x]}</td>
</tr>
</tbody>
</table>

Under TSO: Forbid \texttt{r1}=1, \texttt{r2}=0

Core 0
- \(x\): Shared
- \(y\): Modified
- \([x]\) $\leftarrow$ 1
- \([y]\) $\leftarrow$ 1

Core 1
- \(x\): Invalid
- \(y\): Invalid
- \(r1\) $\leftarrow$ \([y]\)
- \(r2\) $\leftarrow$ \([x]\)

Prefetch \(x\)

Data (\(x = 0\))

Inv
Motivating Example – “Peekaboo”

- Consider \( mp \) with the livelock-avoidance mechanism:

```
<table>
<thead>
<tr>
<th>Core 0</th>
<th>Core 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>(i1) St ( x ) ← 1</td>
<td>(i3) Ld ( r1 ) ← ( y )</td>
</tr>
<tr>
<td>(i2) St ( y ) ← 1</td>
<td>(i4) Ld ( r2 ) ← ( x )</td>
</tr>
<tr>
<td>Under TSO: Forbid ( r1 = 1, r2 = 0 )</td>
<td></td>
</tr>
</tbody>
</table>
```

- 
  - Core 0
    - \( x \): Shared
    - \( y \): Modified
    - \([x]\) ← 1
    - \([y]\) ← 1
  - Core 1
    - \( x \): Invalid
    - \( y \): Invalid
    - \( r1 \) ← \([y]\)
    - \( r2 \) ← \([x]\)

  - Prefetch \( x \)
  - Data \((x = 0)\)
  - Inv
  - Inv-Ack
Motivating Example – “Peekaboo”

• Consider mp with the livelock-avoidance mechanism:

<table>
<thead>
<tr>
<th>Core 0</th>
<th>Core 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>(i1) St [x] ← 1</td>
<td>(i3) Ld r1 ← [y]</td>
</tr>
<tr>
<td>(i2) St [y] ← 1</td>
<td>(i4) Ld r2 ← [x]</td>
</tr>
<tr>
<td></td>
<td>Under TSO: Forbid r1=1, r2=0</td>
</tr>
</tbody>
</table>

Core 0
- x: Modified
- y: Modified
- [x] ← 1
- [y] ← 1

Core 1
- x: Invalid
- y: Invalid
- r1 ← [y]
- r2 ← [x]
Motivating Example – “Peekaboo”

• Consider mp with the livelock-avoidance mechanism:

<table>
<thead>
<tr>
<th>Core 0</th>
<th>Core 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>(i1) St [x] ← 1</td>
<td>(i3) Ld r1 ← [y]</td>
</tr>
<tr>
<td>(i2) St [y] ← 1</td>
<td>(i4) Ld r2 ← [x]</td>
</tr>
</tbody>
</table>

Under TSO: Forbid r1=1, r2=0
Motivating Example – “Peekaboo”

- Consider mp with the livelock-avoidance mechanism:

<table>
<thead>
<tr>
<th>Core 0</th>
<th>Core 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>(i_1) St (x) ← 1</td>
<td>(i_3) Ld (r_1) ← (y)</td>
</tr>
<tr>
<td>(i_2) St (y) ← 1</td>
<td>(i_4) Ld (r_2) ← (x)</td>
</tr>
</tbody>
</table>

Under TSO: Forbid \(r_1=1\), \(r_2=0\)

Core 0
- \(x\): Modified
- \(y\): Modified
- \([x]\) ← 1
- \([y]\) ← 1

Core 1
- \(x\): Invalid
- \(y\): Invalid
- \(r_1\) ← \([y]\)
- \(r_2\) ← \([x]\)
Motivating Example – “Peekaboo”

- Consider `mp` with the livelock-avoidance mechanism:

<table>
<thead>
<tr>
<th>Core 0</th>
<th>Core 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>(i1)  St <code>x</code> ← 1</td>
<td>(i3) Ld r1 ← [y]</td>
</tr>
<tr>
<td>(i2)  St <code>y</code> ← 1</td>
<td>(i4) Ld r2 ← [x]</td>
</tr>
<tr>
<td>Under TSO: Forbid r1=1, r2=0</td>
<td></td>
</tr>
</tbody>
</table>

```
x: Modified
y: Shared
```
```
[x] ← 1
[y] ← 1
```

```
x: Invalid
y: Shared
r1 = 1
r2 ← [x]
```

```
Prefetch x
Data (x = 0)
Inv
Inv-Ack
Request y
Data (y = 1)
```
Motivating Example – “Peekaboo”

- Consider `mp` with the livelock-avoidance mechanism:

<table>
<thead>
<tr>
<th>Core 0</th>
<th>Core 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>(i1) <code>St [x] ← 1</code></td>
<td>(i3) <code>Ld r1 ← [y]</code></td>
</tr>
<tr>
<td>(i2) <code>St [y] ← 1</code></td>
<td>(i4) <code>Ld r2 ← [x]</code></td>
</tr>
</tbody>
</table>

Under TSO: Forbid `r1=1`, `r2=0`

- Prefetch `x`
- Data (`x = 0`)
- Inv
- Inv-Ack
- Request `y`
- Data (`y = 1`)

Core 0
- `x: Modified`
- `y: Shared`
- `[x] ← 1`
- `[y] ← 1`

Core 1
- `x: Invalid`
- `y: Shared`
- `r1 = 1`
- `r2 ← [x]`
Motivating Example – “Peekaboo”

• Consider mp with the livelock-avoidance mechanism:

<table>
<thead>
<tr>
<th>Core 0</th>
<th>Core 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>(i1) St $x$ ← 1</td>
<td>(i3) Ld r1 ← $y$</td>
</tr>
<tr>
<td>(i2) St $y$ ← 1</td>
<td>(i4) Ld r2 ← $x$</td>
</tr>
<tr>
<td>Under TSO: Forbid r1=1, r2=0</td>
<td></td>
</tr>
</tbody>
</table>

Core 0
x: Modified
y: Shared

[x] ← 1
[y] ← 1

Core 1
x: Invalid
y: Shared

r1 = 1
r2 = 0
The Coherence-Consistency Interface (CCI)

- SWMR, DVI, No Stale Data + Expected Coherence = Consistency

- CCI = guarantees that coherence protocol provides to rest of microarchitecture + memory ordering guarantees that rest of microarch. expects from coherence protocol
The Coherence-Consistency Interface (CCI)

SWMR, DVI, No Stale Data + Expected Coherence = Consistency

CCI = guarantees that coherence protocol provides to rest of microarchitecture + memory ordering guarantees that rest of microarch. expects from coherence protocol
The Coherence-Consistency Interface (CCI)

- SWMR, DVI, No Stale Data
- Expected Coherence

\[ \text{CCI} = \text{guarantees that coherence protocol provides to rest of microarchitecture} + \text{memory ordering guarantees that rest of microarch. expects from coherence protocol} \]
The Coherence-Consistency Interface (CCI)

SWMR, DVI, No Stale Data + Expected Coherence = Consistency

CCI = guarantees that coherence protocol provides to rest of microarchitecture + memory ordering guarantees that rest of microarch. expects from coherence protocol
The Coherence-Consistency Interface (CCI)

• CCI = guarantees that coherence protocol provides to rest of microarchitecture + memory ordering guarantees that rest of microarch. expects from coherence protocol
The Coherence-Consistency Interface (CCI)

CCI = guarantees that coherence protocol provides to rest of microarchitecture +
memory ordering guarantees that rest of microarch. expects from coherence protocol
The Coherence-Consistency Interface (CCI)

- SWMR, DVI, No Livelock + Expected Coherence = Consistency

- CCI = guarantees that coherence protocol provides to rest of microarchitecture + memory ordering guarantees that rest of microarch. expects from coherence protocol
The Coherence-Consistency Interface (CCI)

- CCI = guarantees that coherence protocol provides to rest of microarchitecture + memory ordering guarantees that rest of microarch. expects from coherence protocol

Consistency Violation!
Our Work: CCICheck
Static CCI-aware consistency verification

Coherence Orderings (SWMR, DVI, etc.)

Microarch spec
Litmus Test

<table>
<thead>
<tr>
<th>Core 0</th>
<th>Core 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>(i1) St [x] ← 1</td>
<td>(i3) Ld r1 ← [y]</td>
</tr>
<tr>
<td>(i2) St [y] ← 1</td>
<td>(i4) Ld r2 ← [x]</td>
</tr>
</tbody>
</table>

Under TSO: Forbid r1=1, r2=0
Our Work: CCICheck
Static CCI-aware consistency verification

Microarch spec
Litmus Test

<table>
<thead>
<tr>
<th>Core 0</th>
<th>Core 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>(i1) St x ← 1</td>
<td>(i3) Ld r1 ← [y]</td>
</tr>
<tr>
<td>(i2) St y ← 1</td>
<td>(i4) Ld r2 ← [x]</td>
</tr>
</tbody>
</table>

Under TSO: Forbid r1=1, r2=0

Microarchitectural happens-before (µhb) graph
Background: PipeCheck

- **Exhaustive enumeration of executions using µhb graphs**
- **Cyclic graph →forbidden by µarch**
- **Acyclic graph →allowed by µarch**

<table>
<thead>
<tr>
<th>Litmus Test</th>
<th>Core 0</th>
<th>Core 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>(i1)</td>
<td>St x ← 1</td>
<td>(i3) Ld r1 ← y</td>
</tr>
<tr>
<td>(i2)</td>
<td>St y ← 1</td>
<td>(i4) Ld r2 ← x</td>
</tr>
<tr>
<td>Under TSO:</td>
<td>Forbid r1=1, r2=0</td>
<td></td>
</tr>
</tbody>
</table>

[Lustig et al. MICRO-47]
Background: PipeCheck

- Exhaustive enumeration of executions using $\mu$hb graphs
- Cyclic graph \rightarrow forbidden by $\mu$arch
- Acyclic graph \rightarrow allowed by $\mu$arch

<table>
<thead>
<tr>
<th>Litmus Test $mp$</th>
<th>Core 0</th>
<th>Core 1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(i1) $St x \leftarrow 1$</td>
<td>(i3) $Ld r1 \leftarrow y$</td>
</tr>
<tr>
<td></td>
<td>(i2) $St y \leftarrow 1$</td>
<td>(i4) $Ld r2 \leftarrow x$</td>
</tr>
<tr>
<td>Under TSO: Forbid $r1=1$, $r2=0$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

[Lustig et al. MICRO-47]
Background: PipeCheck

- Exhaustive enumeration of executions using µhb graphs
- Cyclic graph → forbidden by µarch
- Acyclic graph → allowed by µarch

Prior techniques cannot model CCI events!

<table>
<thead>
<tr>
<th>Core 0</th>
<th>Core 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>(i1) St [x] ← 1</td>
<td>(i3) Ld r1 ← [y]</td>
</tr>
<tr>
<td>(i2) St [y] ← 1</td>
<td>(i4) Ld r2 ← [x]</td>
</tr>
</tbody>
</table>

Under TSO: Forbid r1=1, r2=0

[Lustig et al. MICRO-47]
Modelling CCI Events

• Need to model **per-cache occupancy**
  – Lazy coherence and partial incoherence (e.g. GPUs)

• Need to model **coherence transitions** that relate to consistency (e.g. Peekaboo)
Modelling CCI Events

- Need to model **per-cache occupancy**
  - Lazy coherence and partial incoherence (e.g. GPUs)
- Need to model **coherence transitions** that relate to consistency (e.g. Peekaboo)
ViCL: Value in Cache Lifetime

- 4-tuple:
  
  \((\text{cache\_id}, \text{address}, \text{data\_value}, \text{generation\_id})\)

- \text{cache\_id} and \text{generation\_id} uniquely identify each cache line

- A ViCL 4-tuple maps on to the period of time over which the cache line serves the data value for the address

- ViCLs start at a \textbf{ViCL Create} event and end at a \textbf{ViCL Expire} event
ViCL: Value in Cache Lifetime

• 4-tuple: 
  \[(\text{cache_id}, \text{address}, \text{data_value}, \text{generation_id})\]

• \text{cache_id} and \text{generation_id} uniquely identify each cache line

• A ViCL 4-tuple maps on to the period of time over which the cache line serves the data value for the address

• ViCLs start at a \textbf{ViCL Create} event and end at a \textbf{ViCL Expire} event
ViCL: Value in Cache Lifetime

• 4-tuple:
  
  (cache_id, address, data_value, generation_id)

• cache_id and generation_id uniquely identify each cache line

• A ViCL 4-tuple maps on to the period of time over which the cache line serves the data value for the address

• ViCLs start at a ViCL Create event and end at a ViCL Expire event
ViCL: Value in Cache Lifetime

• 4-tuple:
  \((\text{cache_id}, \text{address}, \text{data_value}, \text{generation_id})\)
• \text{cache_id} and \text{generation_id} uniquely identify each cache line
• A ViCL 4-tuple maps on to the period of time over which the cache line serves the data value for the address
• ViCLs start at a \textbf{ViCL Create} event and end at a \textbf{ViCL Expire} event
ViCL: Value in Cache Lifetime

Conventional **co-mp** timeline \((M = \text{Modified}, S = \text{Shared})\)

<table>
<thead>
<tr>
<th>Core 0</th>
<th>Core 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>(i1) St (x) ← 1</td>
<td>(i3) Ld (r1) ← (x)</td>
</tr>
<tr>
<td>(i2) St (x) ← 2</td>
<td>(i4) Ld (r2) ← (x)</td>
</tr>
<tr>
<td>In TSO: (r1=2, r2=2) Allowed</td>
<td></td>
</tr>
</tbody>
</table>

Litmus Test **co-mp**

```
<table>
<thead>
<tr>
<th>Core 0</th>
<th>Core 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Store</td>
<td>Store</td>
</tr>
<tr>
<td>Req./</td>
<td>Req./</td>
</tr>
<tr>
<td>GetM</td>
<td>Hit</td>
</tr>
<tr>
<td>Hit</td>
<td>Fwd.</td>
</tr>
<tr>
<td>Ack/</td>
<td>Data</td>
</tr>
<tr>
<td>Silent</td>
<td>Evict</td>
</tr>
</tbody>
</table>
```

<table>
<thead>
<tr>
<th>Core 0</th>
<th>Core 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load</td>
<td>Load</td>
</tr>
<tr>
<td>Req./</td>
<td>Req./</td>
</tr>
<tr>
<td>GetS</td>
<td>Hit</td>
</tr>
</tbody>
</table>

\(L1^S\)

\(S\)

\(S\)

\(S\)
ViCL: Value in Cache Lifetime

Conventional **co-mp** timeline \((M = \text{Modified}, S = \text{Shared})\)

<table>
<thead>
<tr>
<th>Core 0</th>
<th>Core 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>(i1) St (x) (\leftarrow) 1</td>
<td>(i3) Ld r1 (\leftarrow) (x)</td>
</tr>
<tr>
<td>(i2) St (x) (\leftarrow) 2</td>
<td>(i4) Ld r2 (\leftarrow) (x)</td>
</tr>
<tr>
<td>In TSO: (r1=2, r2=2) Allowed</td>
<td></td>
</tr>
</tbody>
</table>

Litmus Test **co-mp**

- **Core 0**
  - Store Req./ GetM
  - Data/ Store Hit

- **Core 1**
  - Load Req./ GetS
  - Data Load Hit

- **Shared L2$**
  - \(S\)

- **Core 0 L1$**
  - M
  - M
  - S

- **Core 1 L1$**
  - S

(time)
ViCL: Value in Cache Lifetime

Conventional co-mp timeline \((M = \text{Modified}, S = \text{Shared})\)

Litmus Test co-mp

### Core 0
- **(i1) Store** \(x \leftarrow 1\)
- **(i2) Store** \(x \leftarrow 2\)

In TSO: \(r1=2, r2=2\) Allowed

### Core 1
- **(i3) Load** \(r1 \leftarrow [x]\)
- **(i4) Load** \(r2 \leftarrow [x]\)

---

**Core 0**
- **L1\$**
- **Shared L2\$**

**Core 1**
- **L1\$**

---

**Store**
- **Req./ GetM**
- **Store Hit**

**Data/ Store**
- **Req./ Hit**

**Ack/ Fwd.**
- **Silent**
- **Data**
- **Evict**

**Repl./**
- **Silent**
- **Evict**

**Load**
- **Req./ GetS**
- **Load Hit**
- **Load Hit**

(time)
ViCL: Value in Cache Lifetime

Conventional co-mp timeline \((M = Modified, S = Shared)\)

<table>
<thead>
<tr>
<th>Core 0</th>
<th>Core 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>(i1) St (x) ← 1</td>
<td>(i3) Ld (r1) ← (x)</td>
</tr>
<tr>
<td>(i2) St (x) ← 2</td>
<td>(i4) Ld (r2) ← (x)</td>
</tr>
</tbody>
</table>

Litmus Test co-mp

In TSO: \(r1=2, r2=2\) Allowed

Core 0
- L1$
- Shared L2$

Core 1
- L1$

Store Req./ GetM
- Data/ Store Hit
- Ack/ Fwd. Hit
- Repl./ Silent Data
- Evict

M M S

Load Req./ GetS
- Data Load Hit
- Load Req./ Hit

(time)
ViCL: Value in Cache Lifetime

Conventional **co-mp** timeline (M = Modified, S = Shared)

Litmus Test **co-mp**

<table>
<thead>
<tr>
<th>Core 0</th>
<th>Core 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>(i1) St (x) ← 1</td>
<td>(i3) Ld (r1) ← (x)</td>
</tr>
<tr>
<td>(i2) St (x) ← 2</td>
<td>(i4) Ld (r2) ← (x)</td>
</tr>
</tbody>
</table>

In TSO: \(r1=2, r2=2\) Allowed

Core 0
---
L1$

Shared
---
L2$

Core 1
---
L1$

Load Req./ GetS
---
Data Load Hit
---
Load Req./ Hit
---

ViCL: Value in Cache Lifetime

Conventional **co-mp** timeline (M = Modified, S = Shared)

<table>
<thead>
<tr>
<th>Core 0</th>
<th>Core 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>(i1) St $x$ ← 1</td>
<td>(i3) Ld $r1$ ← $x$</td>
</tr>
<tr>
<td>(i2) St $x$ ← 2</td>
<td>(i4) Ld $r2$ ← $x$</td>
</tr>
</tbody>
</table>

Litmus Test **co-mp**

In TSO: $r1=2$, $r2=2$ Allowed

- **Core 0**
  - Store
  - Req./GetM
  - Data/Store
  - Hit

- **Core 1**
  - Load
  - Req./GetS
  - Data
  - Load
  - Hit

- **Shared**
  - S

- **L1$**
  - M
  - M

- **L2$**
  - S

- **Evict**
  - M

- **Time**
ViCL: Value in Cache Lifetime

Now with ViCLs

Litmus Test **co-mp**

<table>
<thead>
<tr>
<th>Core 0</th>
<th>Core 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>(i1) St [x] ← 1</td>
<td>(i3) Ld r1 ← [x]</td>
</tr>
<tr>
<td>(i2) St [x] ← 2</td>
<td>(i4) Ld r2 ← [x]</td>
</tr>
</tbody>
</table>

In TSO: r1=2, r2=2 Allowed

ViCL Nodes

- C: ViCL Create
- E: ViCL Expire

ViCL (cache id, addr, data, gen. id)
ViCL: Value in Cache Lifetime

Now with ViCLs

<table>
<thead>
<tr>
<th>Core 0</th>
<th>Core 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>(i1) St $x \leftarrow 1$</td>
<td>(i3) Ld r1 $\leftarrow x$</td>
</tr>
<tr>
<td>(i2) St $x \leftarrow 2$</td>
<td>(i4) Ld r2 $\leftarrow x$</td>
</tr>
</tbody>
</table>

In TSO: r1=2, r2=2 Allowed

ViCL Nodes
- C: ViCL Create
- E: ViCL Expire

ViCL (cache id, addr, data, gen. id)
ViCL: Value in Cache Lifetime

Now with ViCLs

Litmus Test co-mp

<table>
<thead>
<tr>
<th>Core 0</th>
<th>Core 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>(i1) St [x] ← 1</td>
<td>(i3) Ld r1 ← [x]</td>
</tr>
<tr>
<td>(i2) St [x] ← 2</td>
<td>(i4) Ld r2 ← [x]</td>
</tr>
</tbody>
</table>

In TSO: r1=2, r2=2 Allowed

ViCL Nodes

ViCL (cache id, addr, data, gen. id)
ViCL: Value in Cache Lifetime

Now with ViCLs

Litmus Test co-mp

<table>
<thead>
<tr>
<th>Core 0</th>
<th>Core 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>(i1) St</td>
<td>(i3) Ld</td>
</tr>
<tr>
<td>(i2) St</td>
<td>(i4) Ld</td>
</tr>
<tr>
<td>x ← 1</td>
<td>r1 ← x</td>
</tr>
<tr>
<td>x ← 2</td>
<td>r2 ← x</td>
</tr>
</tbody>
</table>

In TSO: r1=2, r2=2 Allowed

ViCL Nodes

- C: ViCL Create
- E: ViCL Expire

ViCL (cache id, addr, data, gen. id)
ViCL: Value in Cache Lifetime

Can model requests, downgrades, etc.

Litmus Test co-mp

<table>
<thead>
<tr>
<th>Core 0</th>
<th>Core 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>(i1) St [x] ← 1</td>
<td>(i3) Ld r1 ← [x]</td>
</tr>
<tr>
<td>(i2) St [x] ← 2</td>
<td>(i4) Ld r2 ← [x]</td>
</tr>
</tbody>
</table>

In TSO: r1=2, r2=2 Allowed

Core 0
L1$

Shared
L2$

Core 1
L1$

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Store</td>
<td>Data/</td>
<td>Store</td>
<td>Ack/</td>
<td>Repl./</td>
</tr>
<tr>
<td>Req. /</td>
<td>Req. /</td>
<td>Req. /</td>
<td>Fwd.</td>
<td>Silent</td>
</tr>
<tr>
<td>CAM</td>
<td>Hit</td>
<td>Hit</td>
<td>Data</td>
<td>Evict</td>
</tr>
</tbody>
</table>

(0,x,1,0) → (0,x,2,0) → (1,x,2,1) → (2,x,2,0)

Always Enumerated

C ViCL Create
E ViCL Expire

Enumerated as Needed

R $ Line Request
D $ Line Downgrade

Not Enumerated

(C) ViCL Create
(E) ViCL Expire
(R) $ Line Request
ViCL: Value in Cache Lifetime

Can model requests, downgrades, etc.

Litmus Test \texttt{co-mp}

<table>
<thead>
<tr>
<th>Core 0</th>
<th>Core 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>(i1) St (x) (\leftarrow 1)</td>
<td>(i3) Ld (r1) (\leftarrow x)</td>
</tr>
<tr>
<td>(i2) St (x) (\leftarrow 2)</td>
<td>(i4) Ld (r2) (\leftarrow x)</td>
</tr>
</tbody>
</table>

In TSO: \(r1=2, r2=2\) Allowed
ViCLs in µhb Graphs

- Use pipeline model from PipeCheck, but add ViCL nodes and edges

<table>
<thead>
<tr>
<th>Litmus Test co-mp</th>
<th>Core 0</th>
<th>Core 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>(i1) St[x] ← 1</td>
<td></td>
<td>(i3) Ld r1 ← [x]</td>
</tr>
<tr>
<td>(i2) St[x] ← 2</td>
<td></td>
<td>(i4) Ld r2 ← [x]</td>
</tr>
</tbody>
</table>

In TSO: r1=2, r2=2 Allowed
ViCLs in \( \mu \text{hb} \) Graphs

![Diagram of pipeline stages and ViCLs]

- Use pipeline model from PipeCheck, but add ViCL nodes and edges.

<table>
<thead>
<tr>
<th>Litmus Test co-mp</th>
</tr>
</thead>
<tbody>
<tr>
<td>Core 0</td>
</tr>
<tr>
<td>(i1) St [x] ← 1</td>
</tr>
<tr>
<td>(i2) St [x] ← 2</td>
</tr>
</tbody>
</table>

In TSO: r1=2, r2=2 Allowed
ViCLs in μhb Graphs

- Use pipeline model from PipeCheck, but add ViCL nodes and edges

<table>
<thead>
<tr>
<th>Litmus Test co-mp</th>
<th>Core 0</th>
<th>Core 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>(i1) St</td>
<td>x</td>
<td>← 1</td>
</tr>
<tr>
<td>(i2) St</td>
<td>x</td>
<td>← 2</td>
</tr>
</tbody>
</table>

In TSO: r1=2, r2=2 Allowed
ViCLs in μhb Graphs

- Use pipeline model from PipeCheck, but add ViCL nodes and edges

<table>
<thead>
<tr>
<th>Core 0</th>
<th>Core 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>(i1) St [x] ← 1</td>
<td>(i3) Ld r1 ← [x]</td>
</tr>
<tr>
<td>(i2) St [x] ← 2</td>
<td>(i4) Ld r2 ← [x]</td>
</tr>
</tbody>
</table>

In TSO: r1=2, r2=2 Allowed
ViCLs in µhb Graphs

(i1) FetchStage ->po-> (i2)
(i1) DecodeStage
(i1) ExecuteStage
(i1) MemoryStage
(i1) WritebackStage
(i1) Completed
(i1) L1 ViCL Create
(i1) L1 ViCL Expire
(i1) L2 ViCL Create
(i1) L2 ViCL Expire

Core 0
(i1) St x ← 1
(i2) St x ← 2

Core 1
(i3) Ld r1 ← x
(i4) Ld r2 ← x

Litmus Test co-mp

- Use pipeline model from PipeCheck, but add ViCL nodes and edges

In TSO: r1=2, r2=2 Allowed
ViCLs in $\mu$hb Graphs

- Use pipeline model from PipeCheck, but add ViCL nodes and edges

Litmus Test `co-mp`

<table>
<thead>
<tr>
<th>Core 0</th>
<th>Core 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>(i1) St $x$ ← 1</td>
<td>(i3) Ld r1 ← $x$</td>
</tr>
<tr>
<td>(i2) St $x$ ← 2</td>
<td>(i4) Ld r2 ← $x$</td>
</tr>
</tbody>
</table>

In TSO: r1=2, r2=2 Allowed
CCICheck Toolflow

CCICheck μarch specification
1. Instruction Paths
2. Per-Stage Orderings
3. Constraints for Instr. Paths

Path Enum. → Constraint Satisfaction → Pruning (Cycle Checking) → Compare

Litmus Tests

Pass/ Fail
CCICheck Toolflow

CCICheck μarch specification
1. Instruction Paths
2. Per-Stage Orderings
3. Constraints for Instr. Paths

Litmus Tests

Path Enum.

Constraint Satisfaction

Pruning (Cycle Checking)

Compare

Pass/Fail
CCICheck µarch specification
1. Instruction Paths
2. Per-Stage Orderings
3. Constraints for Instr. Paths

CCICheck Toolflow

Path Enum. -> Constraint Satisfaction -> Pruning (Cycle Checking) -> Compare

Litmus Tests

Pass/Fail
CCICheck Toolflow

**CCICheck μarch specification**
1. Instruction Paths
2. Per-Stage Orderings
3. Constraints for Instr. Paths

Litmus Tests

**Diagram: CCICheck Toolflow**
- Path Enum.
- Constraint Satisfaction
- Pruning (Cycle Checking)
- Compare

Pass/Fail
CCICheck Toolflow

CCICheck μarch specification
1. Instruction Paths
2. Per-Stage Orderings
3. Constraints for Instr. Paths

CCICheck

- Path Enum.
- Constraint Satisfaction
- Pruning (Cycle Checking)
- Compare

Litmus Tests

Pass/Fail
## Path Enumeration

<table>
<thead>
<tr>
<th>Thread</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(i1)</td>
<td>St [x] ← 1</td>
</tr>
<tr>
<td>(i2)</td>
<td>Ld r1 ← [x]</td>
</tr>
<tr>
<td>(i3)</td>
<td>Ld r2 ← [x]</td>
</tr>
</tbody>
</table>

Allowed: r1=1, r2=1

## Constraint Satisfaction
Path Enumeration

<table>
<thead>
<tr>
<th>Thread</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>(i1)</td>
<td>St [x]  ← 1</td>
</tr>
<tr>
<td>(i2)</td>
<td>Ld r1 ← [x]</td>
</tr>
<tr>
<td>(i3)</td>
<td>Ld r2 ← [x]</td>
</tr>
</tbody>
</table>

Allowed: r1=1, r2=1

Constraint Satisfaction
Path Enumeration

<table>
<thead>
<tr>
<th>Thread</th>
</tr>
</thead>
<tbody>
<tr>
<td>(i1)</td>
</tr>
<tr>
<td>(i2)</td>
</tr>
<tr>
<td>(i3)</td>
</tr>
</tbody>
</table>

Allowed: r1=1, r2=1

Constraint Satisfaction
### Path Enumeration

<table>
<thead>
<tr>
<th>Thread</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(i1)</td>
<td>St [x] ← 1</td>
</tr>
<tr>
<td>(i2)</td>
<td>Ld r1 ← [x]</td>
</tr>
<tr>
<td>(i3)</td>
<td>Ld r2 ← [x]</td>
</tr>
</tbody>
</table>

Allowed: r1=1, r2=1

### Constraint Satisfaction

**Unsolvability**

**UsesViCL**

**UsesViCL**

**UsesViCL**
Path Enumeration

Constraint Satisfaction

<table>
<thead>
<tr>
<th>Thread</th>
<th>St</th>
<th>Ld r1</th>
<th>Ld r2</th>
</tr>
</thead>
<tbody>
<tr>
<td>(i1)</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(i2)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(i3)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Allowed: r1=1, r2=1

Unsatisfiable Constraint → Invalid Scenario
### Path Enumeration

<table>
<thead>
<tr>
<th>Thread</th>
<th>Action</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>(i1)</td>
<td>St [x]</td>
<td>← 1</td>
</tr>
<tr>
<td>(i2)</td>
<td>Ld r1</td>
<td>← [x]</td>
</tr>
<tr>
<td>(i3)</td>
<td>Ld r2</td>
<td>← [x]</td>
</tr>
</tbody>
</table>

Allowed: r1=1, r2=1

### Constraint Satisfaction

Unsatisfiable Constraint → Invalid Scenario
<table>
<thead>
<tr>
<th>Thread</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>(i1)</td>
<td>St ([x]) ← 1</td>
</tr>
<tr>
<td>(i2)</td>
<td>Ld r1 ← ([x])</td>
</tr>
<tr>
<td>(i3)</td>
<td>Ld r2 ← ([x])</td>
</tr>
<tr>
<td>Allowed:</td>
<td>r1=1, r2=1</td>
</tr>
</tbody>
</table>

Path Enumeration

Constraint Satisfaction

Unsatisfiable Constraint → Invalid Scenario
**Path Enumeration**

<table>
<thead>
<tr>
<th>Thread</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(i1) St [x] ← 1</td>
<td></td>
</tr>
<tr>
<td>(i2) Ld r1 ← [x]</td>
<td></td>
</tr>
<tr>
<td>(i3) Ld r2 ← [x]</td>
<td></td>
</tr>
</tbody>
</table>

Allowed: r1=1, r2=1

**Constraint Satisfaction**

**Unsatisfiable Constraint → Invalid Scenario**
Path Enumeration

<table>
<thead>
<tr>
<th>Thread</th>
<th>Action</th>
<th>Value</th>
<th>Constraint</th>
</tr>
</thead>
<tbody>
<tr>
<td>(i1)</td>
<td>St [x]</td>
<td>← 1</td>
<td></td>
</tr>
<tr>
<td>(i2)</td>
<td>Ld r1</td>
<td>← [x]</td>
<td></td>
</tr>
<tr>
<td>(i3)</td>
<td>Ld r2</td>
<td>← [x]</td>
<td></td>
</tr>
<tr>
<td>Allowed: r1=1, r2=1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Constraint Satisfaction

Cyclic Graph → Prune

Unsatisfiable Constraint → Invalid Scenario
Case Studies and Results
“Peekaboo”

- Livelock prevention mechanism allows use of stale data
- “Peekaboo” edge completes cycle => outcome forbidden
- Consistency maintained
“Peekaboo”

- Livelock prevention mechanism allows use of stale data
- “Peekaboo” edge completes cycle => outcome forbidden
- Consistency maintained
• Livelock prevention mechanism allows use of stale data

• “Peekaboo” edge completes cycle => outcome forbidden

• Consistency maintained
“Peekaboo”

• Livelock prevention mechanism allows use of stale data
• “Peekaboo” edge completes cycle => outcome forbidden
• Consistency maintained
“Peekaboo”

- Livelock prevention mechanism allows use of stale data
- “Peekaboo” edge completes cycle => outcome forbidden
- Consistency maintained
Partial Incoherence: GPUs

- e.g.: mp with membar fences [Alglave et al. ASPLOS15]
- If fence does not enforce InvCache ordering => no cycle
Partial Incoherence: GPUs

- e.g.: \texttt{mp} with \texttt{membar} fences [Alglave et al. ASPLOS15]
- If fence does not enforce \texttt{InvCache} ordering => no cycle
Partial Incoherence: GPUs

- e.g.: mp with membar fences [Alglave et al. ASPLOS15]
- If fence does not enforce InvCache ordering => no cycle
• Runtimes remain reasonable due to intelligent pruning and unsatisfiable constraint detection
• Subsequent research has used SMT solver-based techniques to run most tests in just seconds! [ASPLOS 2016]
• Runtimes remain reasonable due to intelligent pruning and unsatisfiable constraint detection
• Subsequent research has used SMT solver-based techniques to run most tests in just seconds! [ASPLOS 2016]
Conclusion

• CCI verification is critical to correct operation of complex parallel systems

• **CCICheck**: static CCI-aware microarchitectural consistency verification
  – Partial incoherence (GPUs), lazy coherence, and more!

• µhb graphs, ViCLs, and constraint-based enumeration
  – **Comprehensive** and **intuitive** µarch modelling

• Allows designers to build correct systems with greater ease and confidence
CCICheck: Using $\mu$hb Graphs to Verify the Coherence-Consistency Interface

Yatin A. Manerkar, Daniel Lustig, Michael Pellauer, and Margaret Martonosi

Code available at https://github.com/ymanerka/ccicheck
Lazy Coherence (TSO-CC)

- No eager invalidation of sharers, but “InvCache” edges model the invalidation of a core’s private cache on an L1 miss
- Thus, TSO is maintained
Lazy Coherence (TSO-CC)

- No eager invalidation of sharers, but “InvCache” edges model the invalidation of a core’s private cache on an L1 miss
- Thus, TSO is maintained
Constraint-Based Enumeration

<table>
<thead>
<tr>
<th>Thread</th>
</tr>
</thead>
<tbody>
<tr>
<td>(i1)</td>
</tr>
<tr>
<td>(i2)</td>
</tr>
<tr>
<td>(i3)</td>
</tr>
</tbody>
</table>

Allowed: r1=1, r2=1

- i3 needs a source for its value
- L1 ViCL with same address and data
Constraint-Based Enumeration

<table>
<thead>
<tr>
<th>Thread</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>(i1)</td>
<td>St [x] ← 1</td>
</tr>
<tr>
<td>(i2)</td>
<td>Ld r1 ← [x]</td>
</tr>
<tr>
<td>(i3)</td>
<td>Ld r2 ← [x]</td>
</tr>
</tbody>
</table>

Allowed: r1=1, r2=1

- i3 needs a source for its value
- L1 ViCL with same address and data
Constraint-Based Enumeration

<table>
<thead>
<tr>
<th>Thread</th>
<th>(i1)</th>
<th>St $[x] \leftarrow 1$</th>
<th>(i2)</th>
<th>Ld r1 $\leftarrow [x]$</th>
<th>(i3)</th>
<th>Ld r2 $\leftarrow [x]$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Allowed:</td>
<td>$r1=1$, $r2=1$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- i3 needs a source for its value
- L1 ViCL with same address and data
• i3 needs a source for its value
• L1 ViCL with same address and data
=> Two possibilities enumerated.
Constraint-Based Enumeration

- i3 needs a source for its value
- L1 ViCL with same address and data

=> Two possibilities enumerated.