TransForm: Formally Specifying Transistency Models and Synthesizing Enhanced Litmus Tests

Naorin Hossain  
*Princeton University*

Caroline Trippel  
*Stanford University*

Margaret Martonosi  
*Princeton University*

ISCA 2020
Memory Consistency Models (MCMs) are used to specify legal memory access orderings

- MCMs specify rules for legal values that can be returned when software loads from memory on a shared memory system

**Problem**: cannot reason about program executions impacted by shared virtual memory (VM) state with existing MCM constructs

**This work**: Memory Transistency Models (MTMs) – the superset of MCMs that additionally capture VM-aware ordering specifications
How does VM affect consistency?

Virtual-to-physical address (VA-to-PA) mappings are stored in page table entries (PTEs)...

<table>
<thead>
<tr>
<th>Status bits</th>
<th>VA-to-PA mapping</th>
</tr>
</thead>
<tbody>
<tr>
<td>A D R W</td>
<td>VA x → PA a</td>
</tr>
<tr>
<td>A D R W</td>
<td>VA y → PA b</td>
</tr>
</tbody>
</table>

...and cached in the translation lookaside buffer (TLB).

```
<table>
<thead>
<tr>
<th>VA-to-PA mapping</th>
</tr>
</thead>
<tbody>
<tr>
<td>VA x → PA a</td>
</tr>
<tr>
<td>VA y → PA b</td>
</tr>
</tbody>
</table>
```

When a mapping is changed in the page table, corresponding TLB entries must be invalidated.
Virtual-to-physical address (VA-to-PA) mappings are stored in page table entries (PTEs)...

...and cached in the translation lookaside buffer (TLB).

When a mapping is changed in the page table, corresponding TLB entries must be invalidated.
How does VM affect consistency?

Virtual-to-physical address (VA-to-PA) mappings are stored in page table entries (PTEs)...

When a mapping is changed in the page table, corresponding TLB entries must be invalidated.

**TLB**

<table>
<thead>
<tr>
<th>VA-to-PA mapping</th>
</tr>
</thead>
<tbody>
<tr>
<td>VA x → PA a</td>
</tr>
<tr>
<td>VA y → PA b</td>
</tr>
</tbody>
</table>

Stale mapping access

**AMD Athlon™ 64 and AMD Opteron™ Processor bug:**

INVLP (x86 TLB entry invalidation instruction) fails to invalidate TLB entry in certain cases.
TransForm introduces constructs for ISA-level MTM specification and ELT synthesis

- Formal MTM vocabulary captures system- and hardware-level VM events and interactions with user-facing program instructions
- Enables ISA-level MTM specification
- Enables automated *enhanced litmus test (ELT)* synthesis

```
μarch model  COATCheck  Valid ELT outcome?

handwritten ELT
```

**Diagram:**
- TransForm
  - MTM vocabulary (§III)
  - MCM vocabulary [3] (§II-A)
  - hardware operations (§III-A)
  - system operations (§III-B)
  - ISA-specific MTM (§V-A)
  - ELT synthesis engine (§IV)
  - MCM litmus test synthesis tool [31] (§IV)
  - candidate execution synthesis (§IV-A)
  - pruning for interesting behavior (§IV-B)
  - pruning for unique ELTs (§IV-C)

**Flow:**
- 

**Automated!**
Outline

• Background on ISA-level MCM vocabulary
• Introduction to ISA-level MTM vocabulary
• Automating synthesis of ELTs
• Case Study: an estimated MTM for x86
• Conclusions
ISA-level MCM relations can describe programs and their *candidate executions*

**Program**

<table>
<thead>
<tr>
<th>Core 0</th>
<th>Instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Event = {W_0, R_1, W_2}</td>
</tr>
<tr>
<td></td>
<td>MemoryEvent = {W_0, R_1, W_2}</td>
</tr>
<tr>
<td></td>
<td>Location = {x}</td>
</tr>
<tr>
<td></td>
<td>address = {W_0 \to x, R_1 \to x, W_2 \to x}</td>
</tr>
<tr>
<td></td>
<td>program order (po)</td>
</tr>
<tr>
<td></td>
<td>po = {W_0 \to R_1, R_1 \to W_2}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Candidate execution</th>
<th>Communication (com) relations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Core 0</td>
<td>reads from (rf)</td>
</tr>
<tr>
<td>W_0 x = 1</td>
<td>rf = {W_0 \to R_1}</td>
</tr>
<tr>
<td>R_1 x = 1</td>
<td>coherence order (co)</td>
</tr>
<tr>
<td>W_2 x = 2</td>
<td>co = {W_0 \to W_2}</td>
</tr>
</tbody>
</table>

**Graph**

Core 0

\[ W_0 \xrightarrow{RF} R_1 \xrightarrow{PO} W_2 \xrightarrow{CO} W_2 \]

Accessed data (outcome) symbolically represented by com relations
MCM specifications place constraints on permitted execution behaviors

Consistency predicates constrain candidate execution behavior based on MCM specifications.

Intel x86 processors use the total store order (TSO) memory model (x86-TSO)

Causality – axiom in x86-TSO consistency predicate:
acyclic(rfe + co + fr + ppo + fence)
Augmenting MCMs to include MTM features

<table>
<thead>
<tr>
<th>What MCMs have</th>
<th>What is needed for MTM interactions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Support for static VA-to-PA mappings without aliasing.</td>
<td>Support for VA-to-PA mappings that can be modified during a program’s execution.</td>
</tr>
<tr>
<td>Support for user-level instruction interactions through shared memory.</td>
<td>Support for shared memory interactions between user-level instructions and system- and hardware-level operations.</td>
</tr>
</tbody>
</table>

transistency operations
Outline

• Background on ISA-level MCM vocabulary
• Introduction to ISA-level MTM vocabulary
• Automating synthesis of ELTs
• Case Study: a estimated MTM for x86
• Conclusions
MTM Vocabulary: Hardware-level operations

TransForm supports **page table walks (PT walks)** and **dirty bit updates**

**Ghost instructions**

**PT walk**: loads translation lookaside buffer (TLB) entry

**dirty bit update**: modifies dirty bit in page table entry (PTE)
MTM Vocabulary: Hardware-level operations

TransForm supports **page table walks (PT walks)** and **dirty bit updates**

**Ghost instructions**

- **PT walk**: loads translation lookaside buffer (TLB) entry
- **dirty bit update**: modifies dirty bit in page table entry (PTE)
- **ghost** – relates user-facing MemoryEvent to invoked ghost instructions (numerical subscripts)
MTM Vocabulary: Hardware-level operations

TransForm supports **page table walks (PT walks)** and **dirty bit updates**

**Ghost instructions**

- **PT walk**: loads translation lookaside buffer (TLB) entry
- **dirty bit update**: modifies dirty bit in page table entry (PTE)
- **ghost**: relates user-facing MemoryEvent to invoked ghost instructions (numerical subscripts)
- **rf_ptw**: relates PT walk to user-facing MemoryEvents that access loaded TLB entry
MTM Vocabulary: System-level operations

TransForm supports **address remappings via PTE Writes** and **TLB entry invalidations**

**Support instructions**

**PTE Write**: changes address mapping stored in a PTE for some VA v

```plaintext
C0
W₀ x = 1
W₀₀₀ z = VA x → PA a
R₀₀₀ z = VA x → PA a
R₀₀₁ z = VA y → PA b
R₀₀₂ z = VA x → PA a

C1
Wₚ₀₀₀ v = VA y → PA a
↓po
W₅ y = 2
W₀₀₀₀ v = VA y → PA b
W₀₀₁ v = VA y → PA b
W₀₀₂ v = VA y → PA b
R₀₀₁ v = VA y → PA b
R₀₀₀ v = VA y → PA b
R₀₀₅ v = VA y → PA b
R₀₀₆ v = VA x → PA a
```
MTM Vocabulary: System-level operations

TransForm supports **address remappings via PTE Writes** and **TLB entry invalidations**

**Support instructions**

**PTE Write**: changes address mapping stored in a PTE for some VA $v$

**INVLPG**: invalidates TLB entry

**remap** – relates PTE Writes to invoked INVLPGs
MTM Vocabulary: System-level operations

TransForm supports **address remappings via PTE Writes** and **TLB entry invalidations**

**Support instructions**

**PTE Write**: changes address mapping stored in a PTE for some VA v

**INVLPG**: invalidates TLB entry

**remap** – relates PTE Writes to invoked INVLPGs

**rf_pa** – relates PTE Write for VA v → PA p to user-facing MemoryEvents accessing PA p via VA v

**co_pa**, **fr_pa**, and **fr_va** follow similarly
MTM Vocabulary: System-level operations

TransForm supports **address remappings via PTE Writes and TLB entry invalidations**

**Support instructions**

**PTE Write**: changes address mapping stored in a PTE for some VA v

**INVLPG**: invalidates TLB entry

- **remap** – relates PTE Writes to invoked INVLPGs

- **rf_pa** – relates PTE Write for VA v → PA p to user-facing MemoryEvents accessing PA p via VA v

- **co_pa**, **fr_pa**, and **fr_va** follow similarly
Outline

• Background on ISA-level MCM vocabulary
• Introduction to ISA-level MTM vocabulary
• Automating synthesis of ELTs
• Case Study: an estimated MTM for x86
• Conclusions
From Specification to Test Synthesis

- ELTs include MTM vocabulary and support verification against an MTM spec
  - Goals:
    - Automated
    - Interesting and minimal ("Spanning set")
    - Deduplicated
    - Comprehensive (to a bound)
TransForm’s synthesis engine starts by synthesizing all possible candidate executions up to a bound.

TransForm Synthesis Engine (§IV)

Spanning Set Pruning (§IV-B)
Interesting ELT criteria (§IV-B):
1. #Write > 0
2. Forbidden by MTM
+ Minimality criterion (§IV-B)

Unique ELT Pruning (§IV-C)

Unique ELT suite

MTM spec in Alloy (§V-A)
Synthesis bound
Relation placement rules (§IV-A)

Candidate Execution Synthesis (§IV-A)

Candidate Executions (Alloy)

Relaxation rules (§IV-B)

Pruned Candidate Executions (XML)

TransForm’s synthesis engine starts by synthesizing all possible candidate executions up to a bound.
Candidate executions are pruned for interesting ELT behaviors and checked for minimality


Candidate Executions (Alloy)

- C0
  - R0 x = 1
  - W2 x = 1
  - po↓
  - R1 x = 0
- C1
  - W0 x = 1
  - po↓
  - W2 x = 1
  - po↓
  - R1 y = 2
  - R1 x = 0
  - R3 y = 1

- C0
  - R0 x = 1
  - No Writes
  - R1 x = 0
- C1
  - W0 x = 1
  - po↓
  - R1 y = 2
  - R1 x = 0
  - R3 x = 1

- C0
  - W0 x = 1
  - po↓
  - R1 y = 2
  - R1 x = 0
  - R3 x = 1
  - rf

- C1
  - R0 x = 1
  - po↓
  - R1 y = 2
  - R1 x = 0
  - R3 x = 1
  - rf

No Writes Permitted by MTM
Unique ELTs are found by deduplicating synthesized ELTs with a post-processing script.

MTM spec in Alloy (§V-A)

Synthesis bound

Relation placement rules (§IV-A)

TransForm Synthesis Engine (§IV)

Candidate Execution Synthesis (§IV-A)

Candidate Executions (Alloy)

Spanning Set Pruning (§IV-B)

Interesting ELT criteria (§IV-B): 1. #Write > 0 2. Forbidden by MTM

+ Minimality criterion (§IV-B)

Relaxation rules (§IV-B)

Pruned Candidate Executions (XML)

Unique ELT Pruning (§IV-C)

Unique ELT suite

MTM spec in Alloy (§V-A)

MTM spec in Alloy (§V-A)

Synthesis bound

Relation placement rules (§IV-A)

TransForm Synthesis Engine (§IV)

Candidate Execution Synthesis (§IV-A)

Candidate Executions (Alloy)

Spanning Set Pruning (§IV-B)

Interesting ELT criteria (§IV-B): 1. #Write > 0 2. Forbidden by MTM

+ Minimality criterion (§IV-B)

Relaxation rules (§IV-B)

Pruned Candidate Executions (XML)

Unique ELT Pruning (§IV-C)

Unique ELT suite

C0

R0 x = 1
po↓
R1 x = 0

C1

W2 x = 1

C0

R0 x = 1
po↓
R1 x = 0

C0

W0 x = 1
R1 y = 2
R3 x = 1

C1

W2 y = 2
↓po
R2 x = 0

C0

W0 x =

C1

x = 1
o↓
R2 x = 0

C0

R0 x = 1
po↓
R1 x = 0

C1

W2 x = 1
↓po
R3 x = 1

...
Outline

• Background on ISA-level MCM vocabulary
• Introduction to ISA-level MTM vocabulary
• Automating synthesis of ELTs
• Case Study: an estimated MTM for x86
• Conclusions
x86t_elt transistency predicates are composed of TSO axioms and new transistency-specific axioms

- **x86t_elt**: an approximate x86 transistency model based on prior work and publicly available documentation

- **x86-TSO**: sc_per_loc, rmw_atomicity, causality

  **invlpg (required)**
  acyclic[fr_va + remap + ^po]

  **tlb_causality (auxiliary)**
  acyclic[ptw_source + com]
A per-axiom suite was synthesized for each `x86t_elt` axiom

103 total unique ELTs!
(98 for hardware verification/validation, 5 for diagnosing TLB implementation bugs)
The synthesized x86t_elt suite consisted of all relevant ELTs from prior work (up to the bound) and more

• 21 of 22 relevant ELTs from prior work synthesized
  • 6 ELTs synthesized verbatim  map to 3 ELT programs in x86t_elt suite
  • 15 ELTs can be reduced to a minimal ELT that is synthesized
  • 1 ELT requires a higher bound for synthesis

• 3 ELTs from prior work, 100 new ELTs
Conclusions

• **TransForm**: framework for formal specification of MTMs and ELT synthesis

• Enables modern ISAs to have a formal specification that includes VM

• Offers systems programmers and hardware designers a stronger opportunity for verification of full systems

• **Future work:**
  • Empirical x86 processor testing
  • RISC-V MTM specification

• Available at: [https://github.com/naorinh/TransForm](https://github.com/naorinh/TransForm)
TransForm: Formally Specifying Transistency Models and Synthesizing Enhanced Litmus Tests

Naorin Hossain  
Princeton University

Caroline Trippel  
Stanford University

Margaret Martonosi  
Princeton University

ISCA 2020

https://github.com/naorinh/TransForm