PROGRAM VERIFICATION
UNDER
FORMALIZED MEMORY CONSISTENT MODELS

TATSUYA ABE
RIKEN AICS
(JOINT WORK WITH TOSHIYUKI MAEDA)

SLACS / NSA
MAY 26, 2014
Q. What is Memory Consistent Model (MCM)?

A. MCM is a rule to share a memory among multiple threads.
There exist many MCMs

Thread Thread Thread Thread

Shared memory

We have to understand such MCMs since the MCMs are rules. But,

There exist non-intuitive MCMs!
An example of curious executions

Under Itanium MCM,

Memory: \[ x \] == \[ y \] == 0

Thread 1
\[ x \] = 1;
\[ y \] = \[ r1 \];
\[ y \] = \[ r1 \];

Thread 2
\[ r2 \] = \[ y \];
\[ r3 \] = \[ x \];

\[ r2 \] == 1 && \[ r3 \] == 0 is allowed!
Reordering under Itanium MCM

Under Itanium MCM,

Memory: \([x] == [y] == 0\)

Thread 1

\([x] = 1;\)
\(r1 = [x];\)
\([y] = r1;\)

Thread 2

\(r3 = [x];\)
\(r2 = [y];\)

Since \(r2 == 1\) and \(r3 == 0\) can be reordered,

\(r2 == 1 \&\& r3 == 0\) is allowed!
Speculative behaviors under UPC MCM

Thread 1
r1 = [x];
[x] = 2;

Thread 2
r2 = [x];
[x] = 1;

r1 == 1 && r2 == 2 is allowed!
Speculative behaviors under UPC MCM

Thread 1
r1 = [x];
[x] = 2;

r1 == 1 && r2 == 2 is allowed!

Thread 2
r2 = [x];
[x] = 1;

speculate: [x] = 1
r1 = [x];
[x] = 2;

speculate: [x] = 2
r2 = [x];
[x] = 1;
Reordering depends on an MCM

Can

\[
\begin{align*}
  r2 &= [y]; \\
  r3 &= [x];
\end{align*}
\]

be reordered?

- Sequential consistency: No
- Total Store Ordering: No
- Partial Store Ordering: Yes
- Itanium MCM: Yes
- UPC MCM: No
Our approach to handle various MCMs simultaneously

1. Give a general model (called base model),

2. define an MCM as a constraint on base model,

3. develop a model checker generator

\begin{center}
\begin{tikzpicture}
\node[draw] {an MCM $\Psi$} edge[->] node[above] {model checker generator} \node[draw] {a program $P$} edge[->] node[above] {model checker for $P$} \node[draw] {a verified property $\varphi$} edge[->] node[above] {Does $P$ have $\varphi$ under $\Psi$?}
\end{tikzpicture}
\end{center}
Threads have their own memories, and read-from/write-to a shared memory is simulated by communication among them.
Each thread has its own base model (at the previous slide) for a speculation.
How to define an MCM

\[ \text{Issue}_1 \text{ Rflct}_1 \text{ Issue}_2 \text{ Rflct}_2 \text{ Issue}_3 \text{ Rflct}_3 \ldots \]

Each pair can be reordered on base model.

✓ \[ \text{Issue}_1 \text{ Issue}_2 \text{ Rflct}_1 \text{ Rflct}_2 \text{ Issue}_3 \text{ Rflct}_3 \ldots \]

✓ \[ \text{Issue}_1 \text{ Issue}_2 \text{ Issue}_3 \text{ Rflct}_1 \text{ Rflct}_2 \text{ Rflct}_3 \ldots \]

✓ ... 

MCMs are constraints for base model.

✓ \[ \text{Issue}_1 \text{ Issue}_2 \text{ Rflct}_1 \text{ Rflct}_2 \text{ Issue}_3 \text{ Rflct}_3 \ldots \]

× \[ \text{Issue}_1 \text{ Issue}_2 \text{ Issue}_3 \text{ Rflct}_1 \text{ Rflct}_2 \text{ Rflct}_3 \ldots \]

...
Formal definition of MCMs

Define an MCM as a set of formulas in mathematical logic.

Definition of formula. A formula is a combination of atomic formulas by logical connectives $\neg$ (negation), $\supset$ (implication), and $\forall$ (universal quantifier).

Atomic formulas. $o < o'$ ($o'$ must be performed after $o$) where $o$ and $o'$ are either of the following:

- **Issue** $T (i, a)$
  
  $T$’s instruction $i$ with attributes $a$ is issued.

- **Rflct** [$\Rightarrow T$] ($i, a$)
  
  Issue of $i$ is reflected to $T$, i.e., $T$ can observe $i$’s issue.
Acquire and release semantics of Itanium MCM

Any instruction must wait for all reflections of an instruction with an attribute release that is issued before.

\[
\begin{align*}
[x] &= 1: \text{release}; \\
r1 &= [x];
\end{align*}
\]

\(r1 = [x]\) must wait for reflections of \([x] = 1: \text{release}\).

\[
\begin{align*}
\text{Issue } T (i, \{\text{release}\}) &< \text{Issue } T (i', A) \supset \\
\text{Rflct } T (i, \{\text{release}\}) &< \text{Issue } T (i', A)
\end{align*}
\]

We confirmed that it was possible to write Itanium MCMs.
Lock and unlock of UPC MCM

If $T$ locks $x$, then $T'$ cannot lock $x$ until $T$ unlocks $x$.

We confirmed that it was possible to write UPC MCMs.
Implementation: model checker generator

Skip!
Experiments: model checking under MCMs

Skip!
Related work

Relaxed memory consistency model is a hot topic.

[Yang et al. ’05] proposes an operational specification framework UMM, which cannot handle speculative behaviors.

[Saraswat et al. ’07] uses program transformations to reason about it.

[Boudol et al. ’09] uses a process calculus to reason about it.

[Shen et al. ’99] uses term rewriting to reason about it.

...
State explosion

A verification under an MCM suffers from state explosion.

Consider $m$ threads with $n$ instructions. Under an MCM that allows interleavings,

$$\text{there exist } m \cdot nC_n \cdot (m-1)\cdot nC_n \cdots \cdot 2\cdot nC_n \cdot nC_n \text{ execution traces.}$$
Partial order reduction based on a verified property

To check $o < o'$, use time counter,

$ABCD$ means a state $\{t_A = 1, t_B = 2, t_C = 3, t_D = 4\}$, and

$ACBD$ means a state $\{t_A = 1, t_B = 3, t_C = 2, t_D = 4\}$.

Let us use pairs of terms that occur in an MCM.

Assume a MCM is $A < B \supset C < D$.

Then, pairs of terms that occur in an MCM are $\{\langle A, B \rangle, \langle C, D \rangle\}$.

$ABCD$ means a state $\{t_{A < B} = \text{true}, t_{C < D} = \text{true}\}$, and

$ACBD$ means a state $\{t_{A < B} = \text{true}, t_{C < D} = \text{true}\}$, too.
Theorem proving using partial order reduction

Thread 1  Thread 2
store $x \leftarrow 1$; store $y \leftarrow 2$;
flush $x$; flush $y$;
barrier; barrier;
load $r_1 \leftarrow y$ load $r_2 \leftarrow x$

Question. $r_1 = 2 \land r_2 = 1$?
Merging triples in backward searches

\[
\begin{array}{l}
\text{store}^1 x \leftarrow 1 \quad \text{store}^2 y \leftarrow 2 \\
\quad \downarrow \quad \downarrow \\
\text{flush}^1 x \quad \text{flush}^2 y \\
\quad \downarrow \quad \downarrow \\
\text{barrier}^1 \quad \text{barrier}^2 \\
\quad \downarrow \quad \downarrow \\
\text{load}^1 r_1 \leftarrow y \quad \text{load}^2 r_2 \leftarrow x
\end{array}
\Rightarrow
\begin{array}{l}
\Pi_1(G_1) \quad \{ y = 2 \land r_2 = 1 \} \quad \{ \text{load}^1 r_1 \leftarrow y \} \quad \{ r_1 = 2 \land r_2 = 1 \} \\
\quad \{ ? \} \quad G \quad \{ r_1 = 2 \land r_2 = 1 \}
\end{array}
\]

\[
\begin{array}{l}
\Pi_2(G_2) \quad \{ r_1 = 2 \land x = 1 \} \quad \{ \text{load}^2 r_2 \leftarrow x \} \quad \{ r_1 = 2 \land r_2 = 1 \} \\
\quad \{ ? \} \quad G \quad \{ r_1 = 2 \land r_2 = 1 \}
\end{array}
\]

\[
\begin{array}{l}
\Pi'(G') \quad \{ y = 2 \land x = 1 \} \quad \{ \text{load}^2 r_2 \leftarrow x \} \quad \{ y = 2 \land r_2 = 1 \} \\
\quad \{ ? \} \quad G_1 \quad \{ y = 2 \land r_2 = 1 \}
\end{array}
\]

\[
\begin{array}{l}
\Pi'(G') \quad \{ y = 2 \land x = 1 \} \quad \{ \text{load}^1 r_1 \leftarrow y \} \quad \{ r_1 = 2 \land x = 1 \} \\
\quad \{ ? \} \quad G_2 \quad \{ r_1 = 2 \land x = 1 \}
\end{array}
\]
Semantics of programs with shared memories

\[
\text{load}^i r \leftarrow x, \langle \sigma, s \rangle \downarrow \langle \sigma[r := \langle x \rangle_{\sigma}], s \rangle
\]

\[
\text{store}^i x \leftarrow e, \langle \sigma, s \rangle \downarrow \langle \sigma, s[x := \langle e \rangle_{\sigma}] \rangle
\]

\[
G, \langle \sigma, s \rangle \downarrow \langle \sigma', s' \rangle \quad \langle \sigma', s' \rangle \Rightarrow \langle \sigma'', s'' \rangle
\]

\[
G, \langle \sigma, s \rangle \downarrow \langle \sigma'', s'' \rangle
\]

The $\Rightarrow$ on $\langle \sigma, s \rangle$ is defined as the smallest reflexive and transitive closure that contains $\langle \sigma, s \rangle \Rightarrow \langle \sigma[s \uparrow \{x\}], s \setminus \{x\} \rangle$.

\[
G \setminus \{C\}, \langle \sigma, s \rangle \downarrow \langle \sigma', s' \rangle \quad \{C\}, \langle \sigma', s' \rangle \downarrow \langle \sigma'', s'' \rangle
\]

\[
G, \langle \sigma, s \rangle \downarrow \langle \sigma'', s'' \rangle \quad C \text{ is not } ...
\]
Hoare logic for dependence graphs

Define a dependence graph from a program and an MCM.

\[ E ::= \text{r} | \text{x} | \overline{\text{x}} \]

\[ \Phi ::= E = E | E \leq E | \neg \Phi | \Phi \supset \Phi | \forall r. \Phi | \forall x. \Phi | \forall \overline{x}. \Phi \]

\[ \{[x/r]\Phi\} \text{load}^i r \leftarrow x \{\Phi\} \]

\[ \{[e/\overline{x}] (\Phi \land [e/x]\Phi)\} \text{store}^i x \leftarrow e \{\Phi\} \]

\[ \forall C \in L(G) \quad \{\Phi\} G \setminus C \{\gamma\} \quad \{\gamma\} C \{\psi\} \]

\[ \{\Phi\} G \{\psi\} \]

It is sound and relatively complete to the semantics.
Related work


[Jones 1981] gives a compositional Hoare logic by using the so-called rely/guarantee method.

[O’Hearn 2007] gives a separation logic for concurrent programs with shared memories.

[Kojima & Igarashi 2013] gives a Hoare logic for Single Instruction Multiple Data (SIMD) programs.
Summary

- Propose a base model on which we discuss MCMs,
- define a set of formulas to describe MCMs,
  - confirm possible to write Itanium and UPC MCMs.
- develop a model checker generator that takes an MCM, and
- demonstrate some experiments.

- Give semantics of programs with shared memories,
- define MCMs as translations from programs into graphs, and
- give sound and relatively complete Hoare logic for graphs.