Verification of a lazy cache coherence protocol against a weak memory model

Christopher J. Banks, Marco Elver, Ruth Hoffmann, Susmit Sarkar, Paul Jackson, Vijay Nagarajan

THE UNIVERSITY *of* EDINBURGH
informatics

University of
St Andrews

FMCAD, October 2017

Introduction

In the paper we:

- verify a *lazy, consistency-directed* cache coherence protocol (TSO-CC);
- against its *memory consistency model* (TSO);
- for an unbounded number of processors.

In this talk I will:

- introduce (the basics of) TSO-CC;
- explain the verification problem;
- show how we solve the problem.

Motivation

Why do this?

- Lazy protocols are gathering more and more interest,
- because they take advantage of weaker memory models.
- Here to stay. Need to be verified.

- Traditional protocols have been verified in isolation,
- we aim for a more holistic verification style,
- tying the knot between protocol and memory model.

Consistency & Coherence

Memory Consistency

- Contract between programmer and hardware

$\Rightarrow$ Specifies when memory operations become visible to other processors.

- e.g. SC, TSO, RC,...

Cache Coherence

- Implements propagation of memory operations (protocol)

$\Rightarrow$ Make memory operations visible according to ordering rules of target *memory consistency model*.

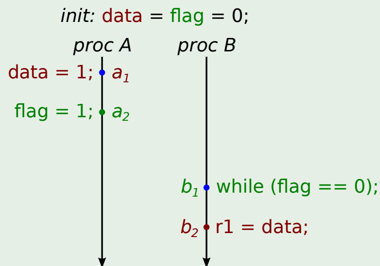
TSO-CC and lazy protocols

TSO-CC:

1. is a lazy cache coherence protocol;
2. takes advantage of weak memory model, TSO;
3. scale to large numbers of cores.

1. Does not invalidate shared locations immediately on a write;
2. because TSO does not require $w \rightarrow r$ ordering;
3. therefore, shared cache-lines are not tracked (storage).

TSO-CC example



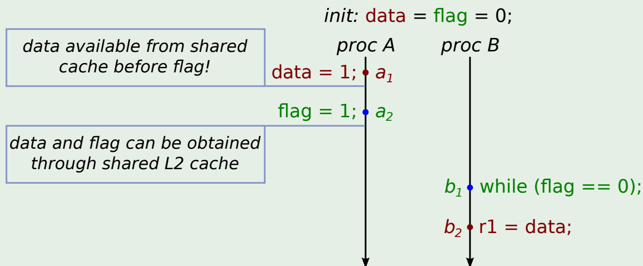
TSO-CC example

init: data = flag = 0;

proc A	*proc B*
data = 1; a_1	
flag = 1; a_2	
	b_1 while (flag == 0);
	b_2 r1 = data;

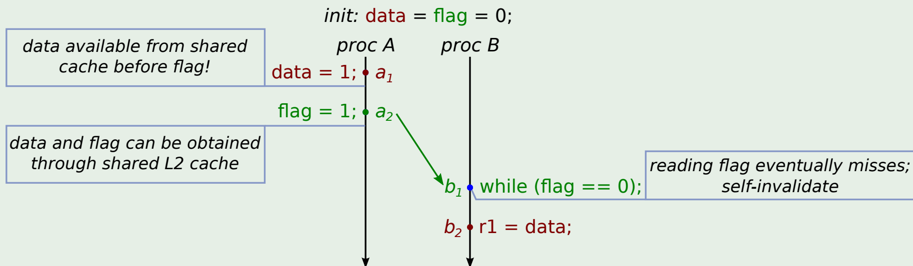
data available from shared cache before flag!

TSO-CC example



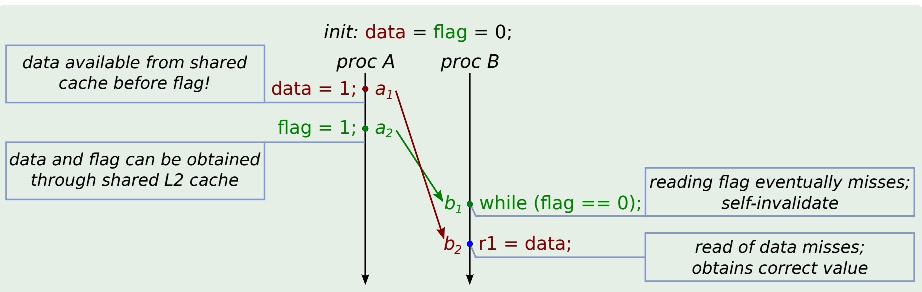
- $w \to w$ order: writes propagate in program order.

TSO-CC example



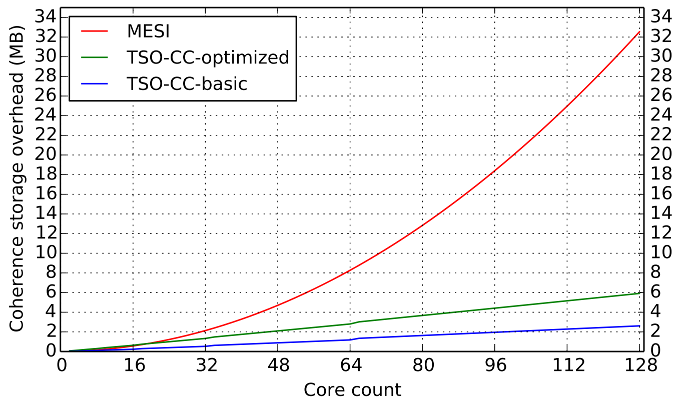
- $w \rightarrow w$ order: writes propagate in program order.
- Relax $w \rightarrow r$ order, but ensure write-propagation.
- Lazy (pull-based updates) vs. eager (push-based).

TSO-CC example



- $w \rightarrow w$ order: writes propagate in program order.
- Relax $w \rightarrow r$ order, but ensure write-propagation.
- Lazy (pull-based updates) vs. eager (push-based).
- $r \rightarrow r$ order: on miss, self invalidate all shared lines.

Storage overheads



Verification

Verification of *eager* protocols

- tends to rely on protocol-specific invariants,
- chiefly, Single Writer Multiple Reader (SWMR),
  - a memory location is either cached for *writing* at *one* cache or cached only for *reading* at zero to *many* caches,
- easily expressed in a model checker.

Verification

Verification of *eager* protocols

- tends to rely on protocol-specific invariants,
- chiefly, Single Writer Multiple Reader (SWMR),
  - a memory location is either cached for *writing* at *one* cache or cached only for *reading* at zero to *many* caches,
- easily expressed in a model checker.

For TSO-CC:

- this sort of invariant does not apply.

Verification

Verification of *eager* protocols

- tends to rely on protocol-specific invariants,
- chiefly, Single Writer Multiple Reader (SWMR),
  - a memory location is either cached for *writing* at *one* cache or cached only for *reading* at zero to *many* caches,
- easily expressed in a model checker.

For TSO-CC:

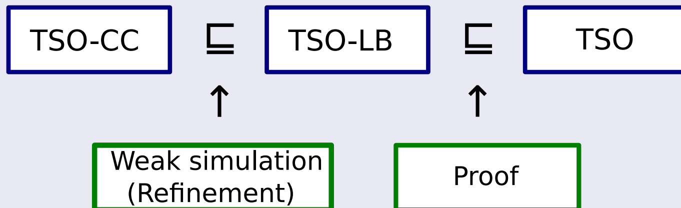
- this sort of invariant does not apply.

So, our approach:

- is to verify protocol satisfies the *memory consistency model*, TSO.

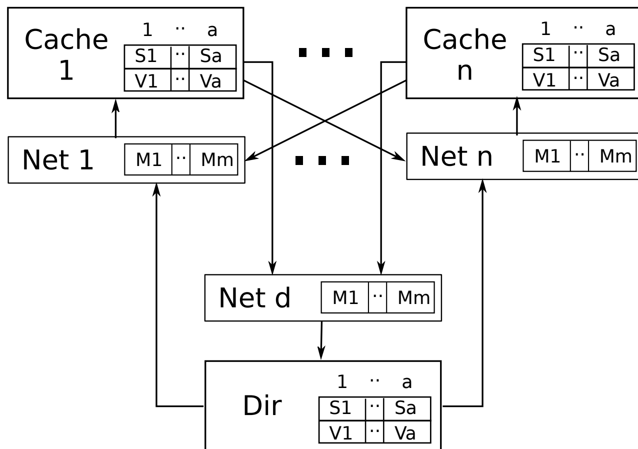
Verification

Overview:



- TSO-CC: implemented in the Murϕ model checker;
- TSO-LB: a finite-state operational model which abstracts pull-based self-invalidation, satisfies TSO;
- Weak simulation (refinement) shown using the model checker;
- Parameterised for n cores.

Modelling TSO-CC



Modelling TSO-CC

Murϕ example rules:

$$c[a].\texttt{state} = \texttt{I} \implies \texttt{SendGetS}(c, Dir, a); \quad c[a].\texttt{state} := WS \qquad \text{(Read I)}$$

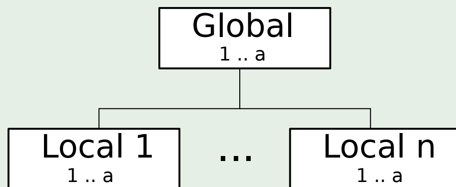
$$c[a].\texttt{state} = \texttt{E} \implies c[a].\texttt{val} := v \quad c[a].\texttt{state} := M; \qquad \text{(Write E)}$$

TSO-LB operational model

- Existing TSO operational models take a store-buffering approach and require unbounded buffers.
- Model checking requires us to bound buffers (finite state space).
- Pull-based self-invalidation and eventual propagation more naturally maps to a load-buffering style.

- Hence TSO-LB.
- Simple model, but captures the laziness in TSO-CC and satisfies TSO.

TSO-LB operational model



$$\frac{\texttt{local}_q(p)(a) = v}{q \xrightarrow{\text{Read}(p,a,v)} q}\ \textsc{Read}$$

$$\frac{}{q \xrightarrow{\text{Write}(p,a,v)} \langle \texttt{local}_q[(p)(a) \mapsto v], \texttt{global}_q[(a) \mapsto v] \rangle}\ \textsc{Write}$$

$$\frac{}{q \xrightarrow{\tau} \langle \texttt{local}_q[(p) \mapsto \texttt{global}_q], \texttt{global}_q \rangle}\ \textsc{Propagate}$$

Verification conditions

TSO-CC $\sqsubseteq$ TSO-LB

- *Weak simulation* relation between TSO-CC and TSO-LB.
- Implies refinement (behavioural inclusion).
- TSO-LB implemented in the model checker.
- Chatterjee et al. (CAV 2002) give the basis for the technique,
- we formalise the condition and show it works in this setting.

TSO-LB $\sqsubseteq$ TSO

- Proof in paper that TSO-LB satisfies TSO.
- (But is actually stronger ($\sqsubset$).)

Weak simulation checking

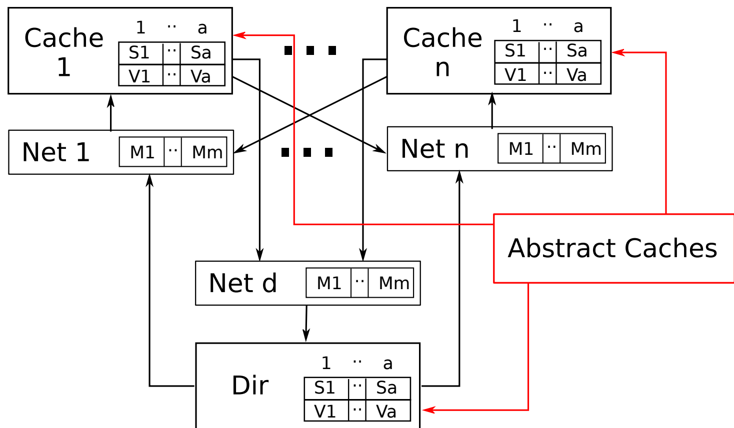
Invocation of TSO-LB from TSO-CC (in the model checker)

$$
\begin{aligned}
c[a].\texttt{state} = \texttt{E} \implies\ & c[a].\texttt{val} := v \\
& c[a].\texttt{state} := M; \\
& \boxed{\texttt{TSOStore}(c, a, v)} && \text{(Write E)} \\
c[a].\texttt{state} = \texttt{S} \implies\ & //\text{do nothing;} \\
& \boxed{\texttt{Assert}(\texttt{TSOVerify}(c, a, c[a].\texttt{val}))} && \text{(Read S)}
\end{aligned}
$$

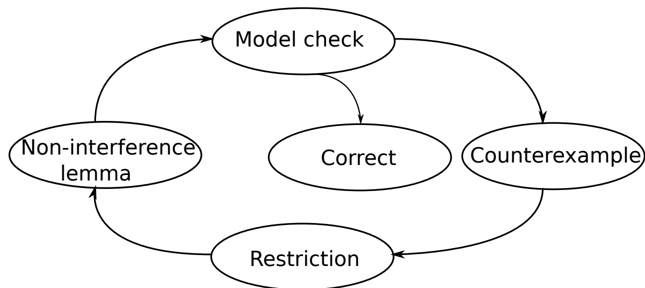
- Weak simulation shown by forcing reads and writes in the protocol to occur in the memory model (silent actions ignored)
- and explicit state space enumeration ensures that the relation is complete.

Parameterisation (unbounded processors)

- Chou et al. (FMCAD 2004)



Parameterisation



- 31 restriction/lemma pairs!
- Tedious, but not difficult.
- Requires knowledge of protocol (analysing counterexample),
- but not deep knowledge of formal methods.
- Potential for automation?

Summary

Contributions:

- verify a lazy cache coherence protocol against its memory consistency model;
- develop a novel operational model, suitable for the purpose, which satisfies TSO;
- formalised a technique for showing refinement by weak simulation in a model checker;
- extend this result to an unbounded number of processors, showing that Chou's parameterised method of verification can be used to show a protocol refines an operational model, not just protocol-specific invariants.

Summary

We believe:

- This is important as protocols like TSO-CC become more prevalent
- and are tied ever more closely to the weaker memory models.

- The technique will apply easily to other protocols targeting TSO
- and possibly other memory models with finite-state operational models (future work).