The Benefits of Duality in Verifying Concurrent Programs under TSO

Parosh Aziz Abdulla

Mohamed Faouzi Atig

Ahmed Bouajjani

Tuan Phong Ngo

1Uppsala University

2IRIF, Université Paris Diderot & IUF

CONCUR 2016
Motivation

Sequential Consistency

- Processes write to/read from shared memory
- Interleaving of the operations
- Program order is persevered for each process

Characteristics

😀 **Simple and intuitive programming model**

😞 **Disallows many hardware optimisations**

👉 **Weak memory models**
Weak Memory Models

**Hardware Optimisations**

- Processors execute instructions **out-of-order**: 😊 Better performance and energy 😞 Non-intuitive behaviours: bugs in racy programs

- **Weak memory model**: captures the semantics of out-of-order execution

**Goal**

- **Efficient** verification technique for checking safety properties
Outline

- Classical TSO semantics
- New semantics (Dual-TSO) allows:
  - Efficient verification
  - Parameterised verification
- Verification under Dual-TSO
- Experimental Results
- Conclusions
TSO - Total Store Order

Widely Used

- Used by Sun SPARCv9
- Current formalisation of Intel x86
TSO - Total Store Order

Widely Used
- Used by Sun SPARCv9
- Current formalisation of Intel x86

Optimise Memory Access
- Memory writes are slow
- Introduce (perfect) store buffers

x = 0
y = 0
TSO - Total Store Order

**Widely Used**
- Used by Sun SPARCv9
- Current formalisation of Intel x86

**Optimise Memory Access**
- Memory writes are slow
- Introduce (perfect) store buffers

**First In First Out (FIFO)**

```
store buffer
x = 0
y = 0
```

**classical semantics**
Classical TSO Semantics

P0: write: x = 1
P0: read: x = 1
P0: read: y = 0

x = 0
y = 0
Classical TSO Semantics

- **P0**: write: \(x = 1\)
- **P0**: read: \(x = 1\)
- **P0**: read: \(y = 0\)

writes to the buffer

- \(x = 0\)
- \(y = 0\)
Classical TSO Semantics

PO: write: $x = 1$

PO: read: $x = 1$

PO: read: $y = 0$

P1

x = 0

y = 0

reads from the buffer
Classical TSO Semantics

PO: write: x = 1
PO: read: x = 1
PO: read: y = 0

read from the memory

x = 0
y = 0
Classical TSO Semantics

PO: write: x = 1
PO: read: x = 1
PO: read: y = 0

P1

x = 1
y = 0

updates to the memory
Verification under TSO is Difficult

while (1)
write: x=1

P0: write: x = 1
P0: write: x = 1
...
P0: write: x = 1
...

x = 0
y = 0
Verification under TSO is Difficult

while (1)
write: x=1
P0: write: x = 1
P0: write: x = 1
P0: write: x = 1
...
...
Verification under TSO is Difficult

while (1)
  write: x=1

PO: write: x = 1
... 
PO: write: x = 1
...
Verification under TSO is Difficult

while (1)
write: x=1

P0: write: x = 1
...
P0: write: x = 1
...

unbounded buffer

infinite state space

x = 0
y = 0
Verification under TSO is Difficult

Existing Methods

• Under approximation
  😞 Can miss bugs

• Over approximation
  😞 Can find spurious bugs

• Exact verification techniques
  😊 Always find real bugs iff they exist
Exact Verification Techniques

Well-Quasi Ordering (WQO) Framework

- Need ordering on state space:
  - Well-quasi ordering
  - **Monotonic** transition system

WQO for TSO

- Sub-word ordering on store buffers?
  - **Not monotonic**!
- WQO cannot be applied easily to TSO
Existing Exact Verification Technique

Monotonic transition system
Single Buffer Model [TACAS’12+13]

merge all buffers

initial message

PO PO

P1 P1

x=0
y=0
Existing Exact Verification Technique

Monotonic transition system
Single Buffer Model [TACAS’12+13]

P0 → merge all buffers
P1

x=0
y=0

PO, P1

viewing pointer
memory snapshot
initial message

= initial message
Existing Exact Verification Technique

Monotonic transition system
Single Buffer Model [TACAS’12+13]

- **P0**: write: \( x = 1 \)
- **P1**: write: \( y = 1 \)

Memory snapshot
Viewing pointer

- **PO**: memory snapshot
- **P1**: memory snapshot

```
PO: x = 0
P1: y = 0
```
Monotonic transition system
Single Buffer Model [TACAS’12+13]

PO: write: x = 1
P1: write: y = 1
...

PO
P1

x,P0

x=1
y=0

x=0
y=0

written variable
ID of writing process

memory snapshot
viewing pointer

Existing Exact Verification Technique
Monotonic transition system
Single Buffer Model [TACAS’12+13]

PO: write: x = 1
P1: write: y = 1

Sub-word relation on the content of the single buffer is a monotonic WQO
Existing Exact Verification Technique

Monotonic transition system
Single Buffer Model [TACAS’12+13]

- memory
- snapshot
- costly
- overhead
- viewing
- pointer
- ID of writing
- process
- cannot be directly
  applied to parameterised
  verification
Parameterised Verification

unbounded number of processes

correctness: lock taken by at most one process

example: mutual exclusion protocols
Our Contribution

**Dual-TSO Model**

- Store buffers are replaced by **load buffers**
- **Equivalent** to classical TSO

**Exact Verification Technique**

- **Efficient** analysis technique based on WQO
- Applicable to **parameterised verification**
Dual-TSO Semantics

Store Buffers ➔ Load Buffers

- Write operations \textbf{immediately update} the memory
- \textbf{Load buffers} contain expected read operations

\begin{align*}
\text{PO} & \quad x,1,\text{self} \\
\text{P1} & \quad x,1,\text{other} \\
\text{load buffer} & \\
\text{self message} & \\
x = 1 \\
y = 0 \\
\text{other message}
\end{align*}
Dual-TSO Semantics

P0: write: \( x = 1 \)
P0: read: \( y = 0 \)

P1

x = 0
y = 0
Dual-TSO Semantics

**PO**: write: $x = 1$

**PO**: read: $y = 0$

- **adds self message**
- **writes to the memory**

$y = 0$

$P_0$

$x,1,self$

$P_1$

$x = 1$

$y = 0$
Dual-TSO Semantics

- **P0**: write: $x = 1$
- **P0**: read: $y = 0$

- Propagates from the memory

- $y = 0$
- $P0$: read: $y = 0$

- $x = 1$
- $P1$: write: $x = 1$

- $x = 1$, $P0$, self
- $x = 1$, $P1$, other
Dual-TSO Semantics

PO: write: x = 1
PO: read: y = 0

y = 0

P0: read: y = 0
P1: x,1,other
P0: write: x = 1
P0: x,1,self
P1: x,1,other

x = 1

propagates from the memory
Dual-TSO Semantics

PO: write: $x = 1$
P0: read: $y = 0$

PO: reads the oldest message

$y,0,\text{other}$

$P0 \rightarrow y,0,\text{other}$

$P1 \rightarrow x,1,\text{other}$

$x,1,\text{other}$

$x = 1$

$y = 0$
Dual-TSO Semantics

PO: write: \( x = 1 \)
PO: read: \( y = 0 \)

reads the oldest message

y,0,other

x,1,other

x = 1
y = 0
The Dual-TSO semantics is equivalent to the TSO semantics with respect to the reachability problem.
Outline

- Classical TSO semantics

- New semantics (Dual-TSO) allows:
  - Efficient verification
  - Parameterised verification

- Verification under Dual-TSO

- Experimental Results

- Conclusions
WQO under Dual-TSO

partition of load buffer

newest self message on x
newest self message on y

Old

New

x,2,self    y,1,self    x,1,other    y,0,self    x,0,other
WQO under Dual-TSO

Extension of sub-word ordering

x,2,self = y,1,self x,1,other = y,0,self x,0,other

x,2,self = y,1,self y,0,self x,0,other
WQO under Dual-TSO

Extension of sub-word ordering
WQO under Dual-TSO

- **Same local states** of processes
- **Same shared memory**
- **Sub-word** relation on load buffers

```
x = 1  
y = 0
```

Diagram showing the states and memory for processes P0 and P1.
WQO under Dual-TSO

- Same local states of processes
- Same shared memory
- Sub-word relation on load buffers
WQO under Dual-TSO

WQO for Dual-TSO

- **Same** local states of processes
- **Same** shared memory
- Sub-word relation on load buffers

\[ x = 1 \]
\[ y = 0 \]
## Dual-TSO vs Single Buffer

<table>
<thead>
<tr>
<th></th>
<th>Dual-TSO</th>
<th>Single Buffer</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>efficient</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>no memory snapshot</td>
<td><strong>NO</strong> memory snapshot</td>
<td>Need memory snapshot</td>
</tr>
<tr>
<td>no viewing pointer, ID of process</td>
<td><strong>No</strong> viewing pointer, ID of process</td>
<td>Need viewing pointers, IDs of processes</td>
</tr>
<tr>
<td>many channels: one channel per process</td>
<td><strong>Many</strong> channels: one channel per process</td>
<td>Only one channel</td>
</tr>
<tr>
<td>buffers have read operations</td>
<td>buffers have read operations</td>
<td>Buffers have write operations</td>
</tr>
<tr>
<td>can be applied to parameterised verification</td>
<td>can be applied to parameterised verification</td>
<td>can be applied to parameterised verification</td>
</tr>
</tbody>
</table>

efficient

no memory snapshot

Need memory snapshot

No viewing pointer, ID of process

Need viewing pointers, IDs of processes

Many channels: one channel per process

Only one channel

Buffers have read operations

Buffers have write operations
Outline

• Classical TSO semantics

• New semantics (Dual-TSO) allows:
  - Efficient verification
  - Parameterised verification

• Verification under Dual-TSO

• Experimental Results

• Conclusions
### Experimental Results

#### Dual-TSO vs Memorax

- **Running time**
- **Memory consumption**

<table>
<thead>
<tr>
<th>Program</th>
<th>#P</th>
<th>Dual-TSO #T</th>
<th>Dual-TSO #C</th>
<th>Memorax #T</th>
<th>Memorax #C</th>
</tr>
</thead>
<tbody>
<tr>
<td>SB</td>
<td>5</td>
<td>0.3</td>
<td>10641</td>
<td>559.7</td>
<td>10515914</td>
</tr>
<tr>
<td>LB</td>
<td>3</td>
<td>0.0</td>
<td>2048</td>
<td>71.4</td>
<td>1499475</td>
</tr>
<tr>
<td>WRC</td>
<td>4</td>
<td>0.0</td>
<td>1507</td>
<td>63.3</td>
<td>1398393</td>
</tr>
<tr>
<td>ISA2</td>
<td>3</td>
<td>0.0</td>
<td>509</td>
<td>21.1</td>
<td>226519</td>
</tr>
<tr>
<td>RWC</td>
<td>5</td>
<td>0.1</td>
<td>4277</td>
<td>61.5</td>
<td>1196988</td>
</tr>
<tr>
<td>W+RWC</td>
<td>4</td>
<td>0.0</td>
<td>1713</td>
<td>83.6</td>
<td>1389009</td>
</tr>
<tr>
<td>IRIW</td>
<td>4</td>
<td>0.0</td>
<td>520</td>
<td>34.4</td>
<td>358057</td>
</tr>
<tr>
<td>Nbw_w_wr</td>
<td>2</td>
<td>0.0</td>
<td>222</td>
<td>10.7</td>
<td>200844</td>
</tr>
<tr>
<td>Sense_rev_bar</td>
<td>2</td>
<td>0.1</td>
<td>1704</td>
<td>0.8</td>
<td>20577</td>
</tr>
<tr>
<td>Dekker</td>
<td>2</td>
<td>0.1</td>
<td>5053</td>
<td>1.1</td>
<td>19788</td>
</tr>
<tr>
<td>Dekker_simple</td>
<td>2</td>
<td>0.0</td>
<td>98</td>
<td>0.0</td>
<td>595</td>
</tr>
<tr>
<td>Peterson</td>
<td>2</td>
<td>0.1</td>
<td>5442</td>
<td>5.2</td>
<td>90301</td>
</tr>
<tr>
<td>Peterson_loop</td>
<td>2</td>
<td>0.2</td>
<td>7632</td>
<td>5.6</td>
<td>100082</td>
</tr>
<tr>
<td>Szymanski</td>
<td>2</td>
<td>0.6</td>
<td>29018</td>
<td>1.0</td>
<td>26003</td>
</tr>
<tr>
<td>MP</td>
<td>4</td>
<td>0.0</td>
<td>883</td>
<td>TO</td>
<td>●</td>
</tr>
<tr>
<td>Ticket_spin_lock</td>
<td>3</td>
<td>0.9</td>
<td>18963</td>
<td>TO</td>
<td>●</td>
</tr>
<tr>
<td>Bakery</td>
<td>2</td>
<td>2.6</td>
<td>82050</td>
<td>TO</td>
<td>●</td>
</tr>
<tr>
<td>Dijkstra</td>
<td>2</td>
<td>0.2</td>
<td>8324</td>
<td>TO</td>
<td>●</td>
</tr>
<tr>
<td>Lamport_fast</td>
<td>3</td>
<td>17.7</td>
<td>292543</td>
<td>TO</td>
<td>●</td>
</tr>
<tr>
<td>Burns</td>
<td>4</td>
<td>124.3</td>
<td>2762578</td>
<td>TO</td>
<td>●</td>
</tr>
</tbody>
</table>

[https://www.it.uu.se/katalog/tuang296/dual-tso](https://www.it.uu.se/katalog/tuang296/dual-tso)
Experimental Results

### Dual-TSO vs Memorax

- Running time
- Memory consumption

#### Standard Benchmarks: Litmus Tests and Mutual Algorithms
Experimental Results

### Dual-TSO vs Memorax

- **Running time**
- **Memory consumption**

<table>
<thead>
<tr>
<th>Program</th>
<th>#P</th>
<th>Dual-TSO</th>
<th>Memorax</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>#T</td>
<td>#C</td>
</tr>
<tr>
<td>SB</td>
<td>5</td>
<td>0.3</td>
<td>10641</td>
</tr>
<tr>
<td>LB</td>
<td>3</td>
<td>0.0</td>
<td>2048</td>
</tr>
<tr>
<td>WRC</td>
<td>4</td>
<td>0.0</td>
<td>1507</td>
</tr>
<tr>
<td>ISA2</td>
<td>3</td>
<td>0.0</td>
<td>509</td>
</tr>
<tr>
<td>RWC</td>
<td>5</td>
<td>0.1</td>
<td>4277</td>
</tr>
<tr>
<td>W+RWC</td>
<td>4</td>
<td>0.0</td>
<td>1713</td>
</tr>
<tr>
<td>IRIW</td>
<td>4</td>
<td>0.0</td>
<td>520</td>
</tr>
<tr>
<td>Nbw_w_wr</td>
<td>2</td>
<td>0.0</td>
<td>222</td>
</tr>
<tr>
<td>Sense_rev_bar</td>
<td>2</td>
<td>0.1</td>
<td>1704</td>
</tr>
<tr>
<td>Dekker</td>
<td>2</td>
<td>0.1</td>
<td>5053</td>
</tr>
<tr>
<td>Dekker_simple</td>
<td>2</td>
<td>0.0</td>
<td>98</td>
</tr>
<tr>
<td>Peterson</td>
<td>2</td>
<td>0.1</td>
<td>5442</td>
</tr>
<tr>
<td>Peterson_loop</td>
<td>2</td>
<td>0.2</td>
<td>7632</td>
</tr>
<tr>
<td>Szymanski</td>
<td>2</td>
<td>0.6</td>
<td>29018</td>
</tr>
<tr>
<td>MP</td>
<td>4</td>
<td>0.0</td>
<td>883</td>
</tr>
<tr>
<td>TicketSpinlock</td>
<td>3</td>
<td>0.9</td>
<td>18963</td>
</tr>
<tr>
<td>Bakery</td>
<td>2</td>
<td>2.6</td>
<td>82050</td>
</tr>
<tr>
<td>Dijkstra</td>
<td>2</td>
<td>0.2</td>
<td>8324</td>
</tr>
<tr>
<td>Lamport_fast</td>
<td>3</td>
<td>17.7</td>
<td>292543</td>
</tr>
<tr>
<td>Burns</td>
<td>4</td>
<td>124.3</td>
<td>2762578</td>
</tr>
</tbody>
</table>

*Note: TO indicates timeout.*
Experimental Results

Dual-TSO vs Memorax

- Running time
- Memory consumption

Dual-TSO is faster and uses less memory in most of examples
Experimental Results
Parameterised Cases

unbounded number of processes

<table>
<thead>
<tr>
<th>Program</th>
<th>Dual-TSO</th>
<th>Dual-TSO</th>
</tr>
</thead>
<tbody>
<tr>
<td>SB</td>
<td>#T: 0.0</td>
<td>#C: 147</td>
</tr>
<tr>
<td>LB</td>
<td>#T: 0.6</td>
<td>#C: 1028</td>
</tr>
<tr>
<td>MP</td>
<td>#T: 0.0</td>
<td>#C: 149</td>
</tr>
<tr>
<td>WRC</td>
<td>#T: 0.8</td>
<td>#C: 618</td>
</tr>
<tr>
<td>ISA2</td>
<td>#T: 4.3</td>
<td>#C: 1539</td>
</tr>
<tr>
<td>RWC</td>
<td>#T: 0.2</td>
<td>#C: 293</td>
</tr>
<tr>
<td>W+RWC</td>
<td>#T: 1.5</td>
<td>#C: 828</td>
</tr>
<tr>
<td>IRIW</td>
<td>#T: 4.6</td>
<td>#C: 648</td>
</tr>
</tbody>
</table>
increasing the number of processes

Experimental Results
Parameterised Cases

<table>
<thead>
<tr>
<th>Program</th>
<th>Dual-TSO</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>#T</td>
</tr>
<tr>
<td>SB</td>
<td>0.0</td>
</tr>
<tr>
<td>LB</td>
<td>0.6</td>
</tr>
<tr>
<td>MP</td>
<td>0.0</td>
</tr>
<tr>
<td>WRC</td>
<td>0.8</td>
</tr>
<tr>
<td>ISA2</td>
<td>4.3</td>
</tr>
<tr>
<td>RWC</td>
<td>0.2</td>
</tr>
<tr>
<td>W+RWC</td>
<td>1.5</td>
</tr>
<tr>
<td>IRIW</td>
<td>4.6</td>
</tr>
</tbody>
</table>
Experimental Results
Parameterised Cases

Dual-TSO is more scalable
Experimental Results
Parameterised Cases

Dual-TSO is more **efficient** and **scalable**

<table>
<thead>
<tr>
<th>Program</th>
<th>Dual-TSO</th>
</tr>
</thead>
<tbody>
<tr>
<td>SB</td>
<td>0.0</td>
</tr>
<tr>
<td>LB</td>
<td>0.6</td>
</tr>
<tr>
<td>MP</td>
<td>0.0</td>
</tr>
<tr>
<td>WRC</td>
<td>0.8</td>
</tr>
<tr>
<td>ISA2</td>
<td>4.3</td>
</tr>
<tr>
<td>RWC</td>
<td>0.2</td>
</tr>
<tr>
<td>W+RWC</td>
<td>1.5</td>
</tr>
<tr>
<td>IRIW</td>
<td>4.6</td>
</tr>
</tbody>
</table>
Summary

Dual-TSO Model

- **Exact (parameterised) reachability method:**
  - **Dual-TSO:** Load buffers instead of store buffers
  - Using well-quasi ordering framework:
    - **Efficient** verification
    - **Parameterised** verification
- Prototype implementation
Future Work

Possible Extension

- Infinite data domain: predicate abstraction
- Apply to more memory models: e.g. PSO
Thank you!

Question?