NON-SPECULATIVE LOAD→LOAD REORDERING IN TSO

Alberto Ros

Universidad de Murcia

October 17th, 2017

OUTLINE

1 BACKGROUND

2 WRITERSBLOCK

3 RESULTS

4 CONCLUSIONS
Program order

- Programmer intuition: instructions execute in the order they appear in the program

Thread 1

$r0 = X; // load
$r1 = Y; // load
PROGRAM ORDER

- Programmer intuition: instructions execute in the order they appear in the program

**THREAD 1**

$r0 = X;  // load
$r1 = Y;  // load

- What happens if the order changes?

**THREAD 1**

$r1 = Y;  // load
$r0 = X;  // load
Program order

- Programmer intuition: instructions execute in the order they appear in the program

**Thread 1**
- \$r0 = X;  // load
- \$r1 = Y;  // load

**Thread 2**
- Y = 1;  // store
- X = 1;  // store

- What happens if the order changes?

**Thread 1**
- \$r1 = Y;  // load
- \$r0 = X;  // load

**Thread 2**
- Y = 1;  // store
- X = 1;  // store
TOTAL STORE ORDER (TSO)

- The memory consistency model defines the behavior of the programs
  - In particular, the behavior of the memory operations: load and store
**Total Store Order (TSO)**

- The memory consistency model defines the behavior of the programs
  - In particular, the behavior of the memory operations: load and store
- x86 processors (Intel, AMD) implement a Total Store Order (TSO)

**TSO Rules**
- load → load
- store → store
- load → store
The memory consistency model defines the behavior of the programs

- In particular, the behavior of the memory operations: load and store

x86 processors (Intel, AMD) implement a Total Store Order (TSO)

**TOTAL STORE ORDER (TSO)**

- load → load
- store → store
- load → store

---

The memory consistency model defines the behavior of the programs

- In particular, the behavior of the memory operations: load and store

x86 processors (Intel, AMD) implement a Total Store Order (TSO)

**TOTAL STORE ORDER (TSO)**

- This talk focuses on the load→load order

---

Possible results under load→load (e.g. TSO)

Initially X=0, Y=0

\[
\begin{align*}
\text{l}x: & \quad r0 = X; \\
\text{l}y: & \quad r1 = Y; \\
\text{s}y: & \quad Y = 1; \\
\text{s}x: & \quad X = 1;
\end{align*}
\]
Possible results under load→load (e.g. TSO)

Initially X=0, Y=0

\[
\begin{align*}
\text{lx}: & \quad r0 = X; \\
\text{ly}: & \quad r1 = Y; \\
\text{sy}: & \quad Y = 1; \\
\text{sx}: & \quad X = 1;
\end{align*}
\]

Six possible interleavings and values for ($r0$, $r1$)

\[
\begin{array}{cccccc}
\text{lx} & \text{ly} & \text{sy} & \text{sx} \\
(0,0) & (0,1) & (0,1) & (0,1) & (0,1) & (1,1)
\end{array}
\]

- (1,0) is not possible if load→load & store→store
RELAXING LOAD $\rightarrow$ LOAD

Initially $X=0$, $Y=0$

- $lx$: $r0 = X$
- $ly$: $r1 = Y$
- $sx$: $X = 1$
- $sy$: $Y = 1$

Six possible interleavings and values for ($r0$, $r1$)

<table>
<thead>
<tr>
<th>ly</th>
<th>lx</th>
<th>sy</th>
<th>sx</th>
</tr>
</thead>
<tbody>
<tr>
<td>lx</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>sx</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>sy</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>sx</td>
<td>ly</td>
</tr>
<tr>
<td></td>
<td>lx</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>sx</td>
</tr>
</tbody>
</table>

- (0,0)
- (0,0)
- (1,0)
- (0,1)
- (1,1)
- (1,1)

(1,0) is possible by relaxing load $\rightarrow$ load
Load→load reordering

- Waiting for a load to finish to start the execution of the next load is very inefficient
LOAD→LOAD REORDERING

- **Waiting** for a load to finish to start the execution of the next load is very **inefficient**
- High-performance processors execute multiple load operations simultaneously
  - Memory level parallelism
- Load operations can execute out of order
- This is correct for **single-core** processors
Executing load operations out of order can relax the load→load order.
Executing load operations out of order can relax the load→load order.
Executing load operations out of order can relax the load→load order.
Executing load operations out of order can relax the load→load order.
Executing load operations out of order can relax the load→load order.
Executing load operations out of order can relax the load→load order.

Program:

ld₁, ld₂, ld₃, ld₄, ld₅, ld₆

Queue (LQ):

ld₁, ld₂, ld₃, ld₄, ld₅, ld₆

Head:

ld₆, ld₅, ld₄, ld₃, ld₂, ld₁

Hit, Miss

Alberto Ros
Informatics Forun, Edinburgh
October 17th, 2017
Executing load operations out of order can relax the load→load order.

Program:
\[
\begin{array}{c}
\text{ld}_1 \\
\text{ld}_2 \\
\text{ld}_3 \\
\text{ld}_4 \\
\text{ld}_5 \\
\text{ld}_6 \\
\end{array} \quad \text{LQ:} \quad \begin{array}{cccccccc}
\text{ld}_6 & \text{ld}_5 & \text{ld}_4 & \text{ld}_3 & \text{ld}_2 & \text{ld}_1 \\
\text{Miss} & \text{Hit} & & & & \\
\end{array} \quad \text{Order:} \quad \begin{array}{c}
\text{ld}_1 \\
\text{ld}_3 \\
\text{ld}_4 \\
\text{ld}_6 \\
\text{ld}_2 \\
\text{ld}_5 \\
\end{array}
\]
In multicore processors reordering loads can affect the expected result.

It is necessary to always maintain the load→load order?
In multicore processors reordering loads can affect the expected result.

It is necessary to always maintain the load→load order?

**Initially X=0, Y=0**

```
$r0 = X;
$r1 = Y;
Y = 1;
X = 1;
/* (1,0) not allowed */
```
In multicore processors reordering loads can affect the expected result.

It is necessary to always maintain the load→load order?

**Initially X=0, Y=0**

$r0 = X; \quad Y = 1;\quad$  
$r1 = Y; \quad X = 1;\quad$  
/* (1,0) not allowed */

**Possible Execution**

$r0 = Y; \quad Y = 1;\quad$  
$r1 = X; \quad X = 1;\quad$  
/* (0, 0) allowed */
Load→load reordering

- In multicore processors reordering loads can affect the expected result.
- It is necessary to always maintain the load→load order?

**Initially** $X=0$, $Y=0$

<table>
<thead>
<tr>
<th>$r0 = X$;</th>
<th>$r1 = Y$;</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Y = 1$;</td>
<td>$X = 1$;</td>
</tr>
</tbody>
</table>

/* (1,0) not allowed */

**Possible Execution**

<table>
<thead>
<tr>
<th>$r0 = Y$;</th>
<th>$r1 = X$;</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Y = 1$;</td>
<td>$X = 1$;</td>
</tr>
</tbody>
</table>

/* (1, 0) not allowed */
In multicore processors reordering loads can affect the expected result.

It is necessary to always maintain the load→load order?

No, if the other cores do not see the reordering.
INITIALLY X=0, Y=0

lx: \$r0 = X; sy: Y = 1;
ly: \$r1 = Y; sx: X = 1;

SIX POSSIBLE INTERLEAVINGS AND VALUES FOR (\$r0, \$r1)

<table>
<thead>
<tr>
<th>ly</th>
<th>ly</th>
<th>ly</th>
<th>ly</th>
<th>ly</th>
<th>ly</th>
</tr>
</thead>
<tbody>
<tr>
<td>lx</td>
<td>sy</td>
<td>sy</td>
<td>sy</td>
<td>sy</td>
<td>sy</td>
</tr>
<tr>
<td>sy</td>
<td>lx</td>
<td>lx</td>
<td>sx</td>
<td>lx</td>
<td>sx</td>
</tr>
</tbody>
</table>

(0,0)  (0,0)  (1,0)  (0,1)  (1,1)  (1,1)

(1,0) is possible by relaxing load \(\rightarrow\) load
Solution: To allow speculative load→load reordering

Some definitions: performed, ordered, source of speculation (SoS)
Solution: To allow speculative load → load reordering

Some definitions: performed, ordered, source of speculation (SoS)
Solution: To allow speculative load→load reordering

Some definitions: performed, ordered, source of speculation (SoS)

Program

ld₁
ld₂
ld₃
ld₄
ld₅
ld₆

LQ

ld₆
ld₅
ld₄
ld₃
ld₂
ld₁

performed

Order

ld₁
ld₃
ld₄
ld₅
ld₂
ld₆
Solution: To allow speculative load→load reordering

Some definitions: performed, ordered, source of speculation (SoS)
Solution: To allow speculative load→load reordering

Some definitions: performed, ordered, source of speculation (SoS)

Program

<table>
<thead>
<tr>
<th>ld₁</th>
<th>ld₂</th>
<th>ld₃</th>
<th>ld₄</th>
<th>ld₅</th>
<th>ld₆</th>
</tr>
</thead>
</table>

LQ

Order

| ld₁ | ld₃ | ld₄ | ld₂ | ld₅ |

M-spec

SoS

head
Solution: To allow speculative load→load reordering

Some definitions: performed, ordered, source of speculation (SoS)
Current multicore avoid incorrect results

- With the help of the cache coherence protocol
SQUASH AND RE-EXECUTE UPON INVALIDATION

Current multicore avoid incorrect results
  - With the help of the cache coherence protocol

1. GetS

LQ \rightarrow \text{L1}

1. GetS

\text{SoS} \rightarrow \text{L1}

\text{L1} \rightarrow \text{LLC}
SQUASH AND RE-EXECUTE UPON INVALIDATION

- Current multicore avoid incorrect results
  - With the help of the cache coherence protocol
Current multicore avoid incorrect results

With the help of the cache coherence protocol

SQUASH AND RE-EXECUTE UPON INVALIDATION
Current multicore avoid incorrect results
  - With the help of the cache coherence protocol

\[ \text{LQ} \quad \begin{array}{c}
  \text{ly} \\
  \text{lx}
\end{array} \]

1. GetS
2. Inv

\[ \text{LLC} \quad \begin{array}{c}
  \text{Ly} \\
  \text{sx} \\
  \text{lx}
\end{array} \]

1. GetXy

\[ \text{L1} \]

\[ \text{L1} \]

\[ \text{GetS_y} \]

\[ \text{Inv} \]
SQUASH AND RE-EXECUTE UPON INVALIDATION

- Current multicore avoid incorrect results
  - With the help of the cache coherence protocol
  - Squashing and re-executing on remote writes

![Diagram showing cache coherence protocol with L1 caches and LLC]

1. Get\(S_x\)
2. Inv\(y\)

\(\text{LQ} \quad \text{ly} \quad \text{lx} \quad \text{L1} \quad \text{L1} \quad \text{LLC}\)

Alberto Ros
Informatics Forum, Edinburgh
October 17th, 2017 12 / 32
**SQUASH AND RE-EXECUTE UPON EVICTIONS**

- What happens when a cache block loaded by an M-spec load is evicted?
  - If the directory stops tracking the block, the M-spec load will not receive an invalidation

![Diagram](image)

Alberto Ros
Informatics Forum, Edinburgh
October 17th, 2017 13 / 32
What happens when a cache block loaded by an M-spec load is evicted?

- If the directory stops tracking the block, the M-spec load will not receive an invalidation.

Solution: Squash and re-execute upon evictions.

This impacts the performance of sequential applications!!!
What happens when a cache block loaded by an M-spec load is evicted?

- If the directory stops tracking the block, the M-spec load will not receive an invalidation

Solution: Squash and re-execute upon evictions

- This impacts the performance of sequential applications!!!
PROBLEMS OF SPECULATION

- Memory-related speculation is the **current solution** to issue loads out of order while keeping the load→load order.
PROBLEMS OF SPECULATION

- Memory-related speculation is the current solution to issue loads out of order while keeping the load→load order
- Why is good?
  - Because squashing is not frequent!
PROBLEMS OF SPECULATION

- Memory-related speculation is the current solution to issue loads out of order while keeping the load→load order.

Why is good?
- Because squashing is not frequent!

Why is bad?
- Because speculative loads hold critical resources (LQ, RoB)
- Because the processor needs to keep continuously the rollback path.
PROBLEMS OF SPECULATION

Memory-related speculation is the current solution to issue loads out of order while keeping the load→load order.

Why is good?
- Because squashing is not frequent!

Why is bad?
- Because speculative loads hold critical resources (LQ, RoB)
- Because the processor needs to keep continuously the rollback path

QUESTION
Can we execute loads out of order without speculation and guaranteeing load→load?
Current multicore processors speculatively execute loads out of order.
If a conflict happens, loads are squashed and re-executed.
WritersBlock achievement

- Current multicore processors speculatively execute loads out of order
- If a conflict happens, loads are squashed and re-executed

WritersBlock

Makes possible removing this speculation, executing loads out of order, and making that an executed load is never squashed because of the consistency model.
How?

- With the help of the cache coherence protocol
How?

- With the help of the cache coherence protocol
With the help of the cache coherence protocol

Alberto Ros
Informatics Forum, Edinburgh
October 17th, 2017
**How?**

- With the help of the cache coherence protocol
With the help of the cache coherence protocol
**How?**

- With the help of the cache coherence protocol
With the help of the cache coherence protocol
- Blocking and delaying the remote write (WritersBlock)
How?

- With the help of the cache coherence protocol
  - Blocking and delaying the remote write (WritersBlock)
  - Until when? Until the load stop being M-spec

$$\begin{array}{c}
\text{LQ} \\
\begin{array}{c}
\text{ly} \\
\text{sx} \\
\text{lx}
\end{array}
\end{array}$$

$$\begin{array}{c}
\begin{array}{c}
\text{ly} \\
\text{sx} \\
\text{lx}
\end{array} \\
(1,0)
\end{array}$$

$$\begin{array}{c}
\begin{array}{c}
\text{ly} \\
\text{sx} \\
\text{lx}
\end{array} \\
(0,0)
\end{array}$$
With the help of the cache coherence protocol
- Blocking and delaying the remote write (WritersBlock)
- Until when? Until the load stop being M-spec
With the help of the cache coherence protocol
- Blocking and **delaying** the remote write (WritersBlock)
- **Until when?** Until the load stop being M-spec
What happens upon an eviction? Do we squash loads?
What happens upon an eviction? Do we squash loads?
No, just need to guarantee that the invalidation will arrive upon a remote write
Evictions

What happens upon an eviction? Do we squash loads?
- No, just need to guarantee that the invalidation will arrive upon a remote write

Solution:
- Clean blocks implement silent evictions

LQ

\[ \text{L1} \]

\[ \text{M-spec} \]

\[ \text{ly, lx} \]

---

**Evictions**

- What happens upon an eviction? Do we squash loads?
  - No, just need to guarantee that the invalidation will arrive upon a remote write

- Solution:
  - Clean blocks implement silent evictions\(^3\)
  - Dirty blocks write back the data but the directory still keeps track

Blocking writes can cause deadlocks

- If $x$ and $y$ are two words within the same cache line

```
<table>
<thead>
<tr>
<th>ly</th>
<th>sy</th>
<th>sx</th>
</tr>
</thead>
<tbody>
<tr>
<td>lx</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$$(1, 0)$$

**Solution:** Blocked writes allow reads to be resolved
Blocking writes can cause deadlocks

- If $x$ and $y$ are two words within the same cache line

![Diagram of cache levels and operations]

- $LQ$: L1 queue
- $L1$: L1 cache
- $LLC$: LLC cache
- $lx$, $ly$, $sx$, $sy$: Cache line entries

**Solution:** Blocked writes allow reads to be resolved
- Blocking writes can cause deadlocks
  - If $x$ and $y$ are two words within the same cache line
  - **Solution**: Blocked writes allow reads to be resolved

- **Diagram**:
  - LQ
  - M-spec
  - L1
  - LLC
  - (1,0)
Blocking writes can cause deadlocks

- If \( x \) and \( y \) are two words within the same cache line
- **Solution**: Blocked writes allow reads to be resolved
Blocking writes can cause deadlocks
- If $x$ and $y$ are two words within the same cache line
- **Solution**: Blocked writes allow reads to be resolved

\[ (1,0) \]
LIVELOCK

- Resolving reads while blocking writes can cause livelock
  - Resolving a read once the data is invalidated will cause a second invalidation
LIVELOCK

- Resolving reads while blocking writes can cause livelock
  - Resolving a read once the data is invalidated will cause a second invalidation

- Solution
  - Reads resolved through WritersBlock must be non-cacheable
  - and cannot resolve M-spec loads (no invalidation will be received)
**DEADLOCK AVOIDANCE**

- **Writers Block** cause writes to be blocked
  - Until a load stop being M-speculative
- Deadlocks will not happen if loads cannot be stopped by a pending write miss
- Other blocking causes:
  - MSHR address occupied by write miss $\Rightarrow$ Duplicate read-write MSHR allocation
  - Full directory/LLC $\Rightarrow$ Non-cacheable loads
  - Atomic Read-Modify-Write $\Rightarrow$ Non-speculative
Case of use: Out-of-order commit

- Out-of-order commit\(^4\) allows the processor to retire instructions from the reorder buffer (RoB) even if they are not at the head.
- It cannot retire instructions that may squash.

**CASE OF USE: OUT-OF-ORDER COMMIT**

- **Out-of-order commit** allows the processor to retire instructions from the reorder buffer (RoB) even if they are not at the head.
- It cannot retire instructions that may squash.
- **WritersBlock** allows the retirement of out-of-order loads.
- Better RoB/LQ usage.

![Diagram](image)

CASE OF USE: OUT-OF-ORDER COMMIT

- Out-of-order commit\(^4\) allows the processor to retire instructions from the reorder buffer (RoB) even if they are not at the head.
- It cannot retire instructions that may squash.
- **WritersBlock** allows the retirement of out-of-order loads.
- Better RoB/LQ usage.

\[^4\text{G. B. Bell and M. H. Lipasti, “Deconstructing Commit”, ISPASS, 2004.}\]
OUTLINE

1 BACKGROUND

2 WRITERS BLOCK

3 RESULTS

4 CONCLUSIONS
SIMULATION ENVIRONMENT

- Simulator: GEMS + OoO processor (TSO)
SIMULATION ENVIRONMENT

- Simulator: GEMS + OoO processor (TSO)
- 16-core multicore
- Silvermont (32-entry RoB), Nehalem (128-entry RoB), and Haswell (192-entry RoB)
**SIMULATION ENVIRONMENT**

- Simulator: GEMS + OoO processor (TSO)
- 16-core multicore
- Silvermont (32-entry RoB), Nehalem (128-entry RoB), and Haswell (192-entry RoB)
- Benchmarks: Splash-3\(^5\) and Parsec-3.0

SIMULATION ENVIRONMENT

- Simulator: GEMS + OoO processor (TSO)
- 16-core multicore
- Silvermont (32-entry RoB), Nehalem (128-entry RoB), and Haswell (192-entry RoB)
- Benchmarks: Splash-3\(^5\) and Parsec-3.0
- Protocols
  - **DIRECTORY**: Directory-based MESI protocol
  - **WRITERSBLOCK**: Extensions to **DIRECTORY**

SIMULATION ENVIRONMENT

- Simulator: GEMS + OoO processor (TSO)
- 16-core multicore
- Silvermont (32-entry RoB), Nehalem (128-entry RoB), and Haswell (192-entry RoB)
- Benchmarks: Splash-3\(^5\) and Parsec-3.0
- Protocols
  - **DIRECTORY**: Directory-based MESI protocol
  - **WritersBlock**: Extensions to **DIRECTORY**
- Commit technique
  - **InOrderCommit**
  - **OoOCommit**

**WritersBlock: Blocked writes**

- Results for **InOrderCommit**
- Normalized to **Directory**

![Bar chart showing blocked writes per Kstores for different benchmarks](chart.png)

- **barnes**
- **blackscholes**
- **bodytrack**
- **canneal**
- **cholesky**
- **dedup**
- **ferret**
- **fluidanimate**
- **fmm**
- **freqmine**
- **lu_cb**
- **lu_ncb**
- **ocean_cp**
- **ocean_ncp**
- **radiosity**
- **radix**
- **raytrace**
- **streamcluster**
- **swaptions**
- **vips**
- **volrend**
- **water_nsquared**
- **water_spatial**
- **x264**

The larger the RoB, the more misspeculations, and the more blocked writes.

Less than 5 blocks per 10,000 stores, on average.
**WritersBlock: Blocked writes**

- Results for **InOrderCommit**
- Normalized to **Directory**
- The larger the RoB, the more misspeculations, and the more blocked writes
WRITERSBLOCK: BLOCKED WRITES

- Results for InOrderCommit
- Normalized to DIRECTORY
- The larger the RoB, the more misspeculations, and the more blocked writes
- Less that 5 blocks per 10,000 stores, on average
WritersBlock: Non-cacheable data

- Results for InOrderCommit
- Normalized to Directory
WRITERSBLOCK: NON-CACHEABLE DATA

- Results for **INOrderCommit**
- Normalized to **DIRECTORY**
- The larger the RoB, the more misspeculations, and the more non-cacheable data
**WritersBlock: Non-cacheable data**

- Results for InOrderCommit
- Normalized to Directory
- The larger the RoB, the more misspeculations, and the more non-cacheable data
- \( \approx 1 \) non-cacheable data per 100,000 loads, on average
WritersBlock: Network traffic

- Results for **InOrderCommit**
- Normalized to **DIRECTORY**
WritersBlock: Network Traffic

- Results for InOrderCommit
- Normalized to Directory
- Network traffic on par
Out-of-order Commit: Processor stalls

- Normalized to Directory + InOrderCommit
Out-of-order Commit: Processor stalls

- Normalized to **Directory + InOrderCommit**
- **InOrderCommit**
- **WritersBlock** does not increase SQ stalls
Out-of-order Commit: Processor stalls

- Normalized to Directory + InOrderCommit
- InOrderCommit
  - WritersBlock does not increase SQ stalls
- OoOCommit
  - WritersBlock reduces RoB and LQ stalls on average respect to Directory
OUT-OF-ORDER COMMIT: EXECUTION TIME

- Normalized to DIRECTORY + InOrderCommit
Out-of-order Commit: Execution time

- Normalized to Directory + InOrderCommit
- InOrderCommit
- WritersBlock does not harm performance on average respect to Directory
Out-of-order Commit: Execution time

- Normalized to Directory + InOrderCommit
- InOrderCommit
  - WritersBlock does not harm performance on average respect to Directory
- OoOCommit
  - WritersBlock improves performance by 11% on average respect to Directory
# Outline

1. **Background**
2. **Writers Block**
3. **Results**
4. **Conclusions**
CONCLUSIONS

With the help of the cache coherence protocol, and without harming performance, we can execute loads out of order and without speculation, and obtaining results as if the loads were executed in order

(LOAD → LOAD)
Conclusions

With the help of the cache coherence protocol, and without harming performance, we can execute loads out of order and without speculation, and obtaining results as if the loads were executed in order ($\text{LOAD} \rightarrow \text{LOAD}$).

Non-speculative loads can increase performance of out-of-order commit by 11%.
NON-SPECULATIVE LOAD→LOAD REORDERING IN TSO

Alberto Ros

Universidad de Murcia

October 17th, 2017