NON-SPECULATIVE STORE COALESING IN TOTAL STORE ORDER

Alberto Ros\textsuperscript{1}  
Stefanos Kaxiras\textsuperscript{2}

\textsuperscript{1}Universidad de Murcia  
arios@ditec.um.es

\textsuperscript{2}Uppsala University  
stefanos.kaxiras@it.uu.se

June 4th, 2018
Goal: Coalescing stores into a single write operation to reduce processor stalls and memory accesses
**Goal:** Coalescing stores into a single write operation to reduce processor stalls and memory accesses

**Problem:** Coalescing can break store order (and programming intuition for x86 processors)
Goal: Coalescing stores into a single write operation to reduce processor stalls and memory accesses

Problem: Coalescing can break store order (and programming intuition for x86 processors)

Solution: Atomicity gives the illusion of order but is hard to scale and costly (e.g., centralized, transactional)
Goal: Coalescing stores into a single write operation to reduce processor stalls and memory accesses

Problem: Coalescing can break store order (and programming intuition for x86 processors)

Solution: Atomicity gives the illusion of order but is hard to scale and costly (e.g., centralized, transactional)

Challenge: To perform multiple stores atomically without speculation (without rollback) and in a distributed manner
Goal: Coalescing stores into a single write operation to reduce processor stalls and memory accesses

Problem: Coalescing can break store order (and programming intuition for x86 processors)

Solution: Atomicity gives the illusion of order but is hard to scale and costly (e.g., centralized, transactional)

Challenge: To perform multiple stores atomically without speculation (without rollback) and in a distributed manner

We present the first solution to this challenge
1 Goal: Coalescing stores

2 Problem: Coalescing breaks store order

3 Solution: Atomicity

4 Challenge: Distributed, non-speculative atomicity

5 Results

6 Conclusions
STORE BUFFER AND TOTAL STORE ORDER

- Proc
- a
- Store Buffer
- ab
- Program Order
- Mem
- Store operations in current x86 processors
STORE BUFFER AND TOTAL STORE ORDER

Store operations in current x86 processors
STORE BUFFER AND TOTAL STORE ORDER

Store operations in current x86 processors
STORE BUFFER AND TOTAL STORE ORDER

First-In First-Out (FIFO)

Store operations in current x86 processors
STORE BUFFER AND TOTAL STORE ORDER

First-In First-Out (FIFO)

Store operations in current x86 processors
STORE BUFFER AND TOTAL STORE ORDER

Total Store Order (TSO)

Store operations in current x86 processors
LIMITATIONS AND THE SOLUTION OF COALESCEING

A. Ros & S. Kaxiras
ISCA’18 @ Los Angeles, CA, USA
June 4th, 2018
LIMITATIONS AND THE SOLUTION OF COALESCEING

- Proc
- Stall
- Store Buffer
- Mem
- a
- b
- c

Individual writes/Coalescing Stores to the same cache line (same color in the example) can coalesce in a single write. No stall, Coalesced writes, Performance and energy improvements.
Limitations and the Solution of Coalescing

A. Ros & S. Kaxiras
ISCA’18 @ Los Angeles, CA, USA
June 4th, 2018
LIMITATIONS AND THE SOLUTION OF COALESCEING

Stores to the same cache line (same color in the example) can coalesce in a single write.
LIMITATIONS AND THE SOLUTION OF COALESCING

Proc

No stall

Coalescing

Store Buffer

Mem

Individual writes can coalesce in a single write.

A. Ros & S. Kaxiras
ISCA’18 @ Los Angeles, CA, USA
June 4th, 2018
LIMITATIONS AND THE SOLUTION OF COALESCING

Proc

Mem

Store Buffer

Coalescing

a, c

Stores to the same cache line (same color in the example)

No stall

Coalesced writes

Performance and energy improvements
Goal: Coalescing stores

Problem: Coalescing breaks store order

Solution: Atomicity

Challenge: Distributed, non-speculative atomicity

Results

Conclusions
THE PROBLEM OF COALESCING STORES

Store order?

First green ⇒ c overtakes b
First blue ⇒ b overtakes a

A new litmus test that captures a TSO violation when breaking store order

A. Ros & S. Kaxiras
ISCA’18 @ Los Angeles, CA, USA
June 4th, 2018
THE PROBLEM OF COALESCEING STORES

Proc

Coalescing Store Buffer

Mem

First green ⇒ c overtakes b

b

a, c

In the paper:
A new litmus test that captures a TSO violation when breaking store order

A. Ros & S. Kaxiras
ISCA’18 @ Los Angeles, CA, USA
June 4th, 2018
THE PROBLEM OF COALEScing STORES

The Problem of Coalescing Stores

Proc

Mem

Coalescing Store Buffer

b
a, c

First green ⇒ c overtakes b
First blue ⇒ b overtakes a

A new litmus test that captures a TSO violation when breaking store order.
THE PROBLEM OF COALESCEING STORES

In the paper: A new litmus test that captures a TSO violation when breaking store order
1. **Goal: Coalescing Stores**

2. **Problem: Coalescing breaks store order**

3. **Solution: Atomicity**

4. **Challenge: Distributed, non-speculative atomicity**

5. **Results**

6. **Conclusions**
**Atomicity: Illusion of Store Order**

- **Proc**
- **Mem**
- **Store Buffer**

*Coalescing forms atomic write groups*

Store order $\Rightarrow$ Atomicity
FORMING ATOMIC WRITE GROUPS

Match younger write to the same cache line
FORMING ATOMIC WRITE GROUPS

Writes to a, b, and c are indivisible
FORMING ATOMIC WRITE GROUPS

Match younger write to the same cache line

Writes to a, b, and c are indivisible

Writes to a, b, c, d, and e are indivisible
FORMING ATOMIC WRITE GROUPS

Match younger write to the same cache line

- Writes to a, b, and c are indivisible
- Writes to a, b, c, d, and e are indivisible
FORMING ATOMIC WRITE GROUPS

Market younger write to the same cache line

Writes to a, b, and c are indivisible

Writes to a, b, c, d, and e are indivisible

A. Ros & S. Kaxiras

ISCA’18 @ Los Angeles, CA, USA

June 4th, 2018
**Known ways to perform writes atomically**

1. Mutual exclusion (TCC *ISCA*’04, BulkSC *ISCA*’07)
2. Transactional (Oklahoma *PDTSA*’93, Store-Wait-Free *ISCA*’07)

![Diagram showing mutual exclusion and transactional methods in concurrency control.](image-url)
KNOWN WAYS TO PERFORM WRITES ATOMICALLY

1. Mutual exclusion (TCC ISCA’04, BulkSC ISCA’07)
2. Transactional (Ocklahoma PDTSA’93, Store-Wait-Free ISCA’07)

Centralized and non-scalable solution
Known ways to perform writes atomically

1. Mutual exclusion (TCC ISCA’04, BulkSC ISCA’07)
2. Transactional (Oklahoma PDTSA’93, Store-Wait-Free ISCA’07)

- Speculation: rollback on conflict
- Canceling memory writes is a costly operation
1. **Goal:** Coalescing stores
2. **Problem:** Coalescing breaks store order
3. **Solution:** Atomicity
4. **Challenge:** Distributed, non-speculative atomicity
5. **Results**
6. **Conclusions**
A new perspective

- Writing atomically a number of cache lines is similar to the problem of acquiring a number of locks in parallel programming.

Dijkstra, "Hierarchical ordering of sequential processes"
- Writing atomically a number of cache lines is similar to the problem of acquiring a number of locks in parallel programming.
- **Deadlock**, if locks are taken in opposite order.
Perform writes following a global order
Deadlock-free considering unlimited resources

1 Dijkstra, “Hierarchical ordering of sequential processes”
LEXICOGRAPHICAL ORDER

- Private caches
- Shared directory

A. Ros & S. Kaxiras
ISCA’18 @ Los Angeles, CA, USA
June 4th, 2018
Write in lexicographical (Lex) order ⇒ physical address
LEXYCOGRAPHICAL ORDER

A write locks the cache line permission (lock bit) in private caches. A “conflict” always happens in their minimum common address. All lock bits reset in bulk.
**Lexicographical Order**

A write locks the cache line permission (lock bit)
LEXICOGRAPHICAL ORDER

Group writes have been ordered ⇒ Proc 1 first

A “conflict” between atomic groups always happens in their minimum common address.

Proc 1

Proc 2

Cache

Wait

Mem

Dir
LEXICOGRAPHICAL ORDER

Write in lexicographical (Lex) order

Group writes have been ordered

Proc 1 first

- Private caches
- Shared directory

A write locks the cache line permission (lock bit)

A “conflict” between atomic groups always happens in their minimum common address

All lock bits reset in bulk

A. Ros & S. Kaxiras

ISCA’18 @ Los Angeles, CA, USA

June 4th, 2018
LEXICOGRAPHICAL ORDER

Write in lexicographical (Lex) order
⇒
physical address
Group writes have been ordered
⇒
Proc 1
first
Proc 2

Cache

Dir

Wait

A write locks the cache line permission (lock bit)
A “conflict” between atomic groups always happens in their minimum common address
All lock bits reset in bulk

A. Ros & S. Kaxiras
ISCA’18 @ Los Angeles, CA, USA
June 4th, 2018
LEXICOGRAPHICAL ORDER

Write in lexicographical (Lex) order
⇒ physical address
Group writes have been ordered
⇒ Proc 1 first

Mem
Cache
Dir
a
b
Wait
a
b

Proc 1

Proc 2

All lock bits reset in bulk

A. Ros & S. Kaxiras
ISCA’18 @ Los Angeles, CA, USA
June 4th, 2018
**LEXICOGRAPHICAL ORDER**

Write in lexicographical (Lex) order

- Proc 1 first

- Proc 2

- Private caches

- Shared directory

A write locks the cache line permission (lock bit)

A "conflict" between atomic groups always happens in their minimun common address

All lock bits reset in bulk

A. Ros & S. Kaxiras

ISCA’18 @ Los Angeles, CA, USA

June 4th, 2018
Resource-Conflict Deadlocks

- Lex order is deadlock-free, assuming unlimited resources
  - But resources are limited
Lex order is deadlock-free, assuming unlimited resources
- But resources are limited
- **Locking** cache lines introduces resource-conflict deadlocks
  ⇒ Need resources to keep all locks simultaneously
RESOURCE-CONFLICT DEADLOCKS

- Lex order is deadlock-free, assuming unlimited resources
  - But resources are limited
- Locking cache lines introduces resource-conflict deadlocks
  ⇒ Need resources to keep all locks simultaneously
- Intra-group: resource deadlocks for a single group

\[
\begin{array}{c}
2 \\
1 \\
\end{array}
\Rightarrow \begin{array}{c}
b \\
a \\
\end{array}
\]
Lex order is deadlock-free, assuming unlimited resources
- But resources are limited
- Locking cache lines introduces resource-conflict deadlocks
  \[\Rightarrow\] Need resources to keep all locks simultaneously

1. Intra-group: resource deadlocks for a single group

2. Inter-group: resource deadlocks for multiple groups
Intra-group conflicts in private resources

- Caches must be able to hold all locked cache lines
  - E.g., if direct-mapped cache
    - and a and b map to the same set
      \[ \Rightarrow \text{deadlock} \]
Caches must be able to hold all locked cache lines
- E.g., if direct-mapped cache
- and a and b map to the same set
  \[ \implies \text{deadlock} \]

Sub-address lex order

\[
\text{rank} = \text{addr}_{\text{line}} \mod (\text{sets}_{\text{cache}} \times \text{assoc}_{\text{cache}})
\]

- Reduces coalescing opportunities
  \[ \implies \text{Addresses of the same rank cannot be in the same atomic group} \]
INTER-GROUP CONFLICTS IN SHARED RESOURCES

The formation of atomic groups with sub-address order prevents different atomic groups from overflowing shared structures.

A. Ros & S. Kaxiras
ISCA’18 @ Los Angeles, CA, USA
June 4th, 2018
INTER-GROUP CONFLICTS IN SHARED RESOURCES

The formation of atomic groups with sub-address order prevents different atomic groups from overflowing shared structures.
The formation of atomic groups with sub-address order prevents different atomic groups from overflowing shared structures.
INTER-GROUP CONFLICTS IN SHARED RESOURCES

The formation of atomic groups with sub-address order prevents different atomic groups from overflowing shared structures.

\[ \text{rank} = \text{addr}_{\text{line}} \% (\text{sets}_{\text{dir}} \times \text{assoc}_{\text{dir}}) \]
**System-Wide Sub-address Lex Order**

**Deadlock free:** \( rank = addr_{line} \% \min(sets_i \times assoc_i) \)

- Sub-address lex order intuition
  - Each rank in an order either has resources or conflicts with the minimum common address when taking the resource
  - A conflict orders the atomic group writes
**SYSTEM-WIDE SUB-ADDRESS LEX ORDER**

**Deadlock free:** \( rank = addr_{line} \mod \min(sets_i \times assoc_i) \)

- **Sub-address lex order intuition**
  - Each rank in an order either **has resources or conflicts** with the minimum common address when taking the resource
  - A conflict orders the atomic group writes
- **Simple implementation** in the store buffer
  - Just stop coalescing on rank conflict
**System-Wide Sub-address Lex Order**

**Deadlock free:** \[ rank = \text{addr}_{\text{line}} \% \min \left( \text{sets}_i \times \text{assoc}_i \right) \]

- Sub-address lex order intuition
  - Each rank in an order either has resources or conflicts with the minimum common address when taking the resource
  - A conflict orders the atomic group writes
- Simple implementation in the store buffer
  - Just stop coalescing on rank conflict
- No significant protocol changes
  - Just request waiting and prefetch nacks
OUTLINE

1. **Goal:** Coalescing stores

2. **Problem:** Coalescing breaks store order

3. **Solution:** Atomicity

4. **Challenge:** Distributed, non-speculative atomicity

5. **Results**

6. **Conclusions**
SIMULATION ENVIRONMENT

- Schemes evaluated:
  - **NSB**: Unified SQ/SB, no coalescing (Intel-like)
  - **LSB**: Split SQ/SB, line coalescing
  - **CSB-TSO**: Split SQ/SB, coalescing, TSO
  - **CSB-RC**: Split SQ/SB, coalescing, release consistency
**SIMULATION ENVIRONMENT**

- Schemes evaluated:
  - **NSB**: Unified SQ/SB, no coalescing (Intel-like)
  - **LSB**: Split SQ/SB, line coalescing
  - **CSB-TSO**: Split SQ/SB, coalescing, TSO
  - **CSB-RC**: Split SQ/SB, coalescing, release consistency

- **GEMS + in-house TSO processor model**
  - 8 out-of-order Haswell-like cores
  - Store queue (SQ) + store buffer (SB): 42 entries
  - Lex order: **512 ranks** (L1 cache: 32KB)

- Benchmarks: Parsec-3.0
**ENERGY CONSUMPTION (L1 & SQ/SB)**

- Normalized to **NSB**
- Reductions of *writes* due to coalescing
- Reductions of *reads* due to hits in the SB (more coalescing)

### Results

<table>
<thead>
<tr>
<th>1. NSB</th>
<th>2. LSB</th>
<th>3. CSB-TSO</th>
<th>4. CSB-RC</th>
</tr>
</thead>
</table>

### Challenges

- CSB-TSO
- 23.3% reduction w.r.t NSB
- CSB-TSO on par to CSB-RC

### Conclusions

A. Ros & S. Kaxiras

**ISCA’18 @ Los Angeles, CA, USA**

June 4th, 2018
**ENERGY CONSUMPTION (L1 & SQ/SB)**

- Normalized to **NSB**
- Reductions of **writes** due to coalescing
- Reductions of **reads** due to hits in the SB (more coalescing)

CSB-TSO **23.3%** reduction w.r.t **NSB**

CSB-TSO on par to **CSB-RC**
**Execution time**

- Normalized to **NSB**
- Improvements due to less processor stalls
Normalized to **NSB**

- Improvements due to less processor stalls

**CSB-TSO** improves **NSB** by **6.2%**

**CSB-TSO** close to **CSB-RC**
1. **Goal:** Coalescing stores

2. **Problem:** Coalescing breaks store order

3. **Solution:** Atomicity

4. **Challenge:** Distributed, non-speculative atomicity

5. **Results**

6. **Conclusions**
First solution to perform writes atomically
⇒ Non-centralized
⇒ Non-speculative
⇒ Deadlock-free
First solution to perform writes atomically
  ⇒ Non-centralized
  ⇒ Non-speculative
  ⇒ Deadlock-free

Thanks to LEX order
  ⇒ Non-deadlocking
  ⇒ Accommodates resource limitations
First solution to perform writes atomically
⇒ Non-centralized
⇒ Non-speculative
⇒ Deadlock-free

Thanks to LEX order
⇒ Non-deadlocking
⇒ Accommodates resource limitations

The result is a simpler, higher performing solution
Non-Speculative Store Coalescing in Total Store Order

Alberto Ros\textsuperscript{1}  \quad Stefanos Kaxiras\textsuperscript{2}

\textsuperscript{1}Universidad de Murcia  
aro@ditec.um.es

\textsuperscript{2}Uppsala University  
stefanos.kaxiras@it.uu.se

June 4th, 2018