BulkSC: Bulk Enforcement of Sequential Consistency

*Luis Ceze*, James Tuck, Pablo Montesinos, Josep Torrellas

University of Illinois at Urbana-Champaign

http://iacoma.cs.uiuc.edu

ISCA 2007, San Diego, CA
Memory Consistency Model
Memory Consistency Model

• Defines the values that a read can return
Memory Consistency Model

- Defines the values that a read can return
- Supported by the hardware
Memory Consistency Model

• Defines the values that a read can return
• Supported by the hardware
• Has major implications on programmability
Memory Consistency Model

• Defines the values that a read can return
• Supported by the hardware
• Has major implications on programmability

Flag = Data = 0

P1
Data = 2000
Flag = 1

P2
while (Flag==0){;}
...
= Data
Memory Consistency Model

- Defines the values that a read can return
- Supported by the hardware
- Has major implications on programmability

```plaintext
Flag = Data = 0

P1
Data = 2000
Flag = 1

P2
while (Flag==0){;}
... = Data
```
Memory Consistency Model

- Defines the values that a read can return
- Supported by the hardware
- Has major implications on programmability

```plaintext
Flag = Data = 0

P1

Data = 2000
Flag = 1

P2

while (Flag==0){;
...
= Data
```
Memory Consistency Model

• Defines the values that a read can return
• Supported by the hardware
• Has major implications on programmability

Affects the whole software stack:
• applications
• OS, libraries, drivers
• compilers
• language semantics

while (Flag == 0) {
    ...
    = Data
    Flag = 1
    ...
    = Data
Sequential Consistency (SC)
Sequential Consistency (SC)
Sequential Consistency (SC)

Memory

P1 st A

P2 st C

P3 ld C

PN ld A

[Lamport’79]
Sequential Consistency (SC)

Per-processor program order: memory operations from individual processors maintain program order

[Lamport’79]
Sequential Consistency (SC)

Per-processor program order: memory operations from individual processors maintain program order

[Lamport’79]
Sequential Consistency (SC)

Per-processor program order: memory operations from individual processors maintain program order

[Network Diagram]

Global Order

P1

P1

Memory

st A

st C

ld C

ld A

st A

ld C

st C

[Network Diagram]

[Lamport’79]
Sequential Consistency (SC)

Per-processor program order: memory operations from individual processors maintain program order

Memory

Global Order

P1

[lamport’79]
Sequential Consistency (SC)

Per-processor program order: memory operations from individual processors maintain program order

Single sequential order: the memory operations from all processors maintain a single sequential order

[Senhadji et al.'79]
Sequential Consistency (SC)

Per-processor program order: memory operations from individual processors maintain program order

Single sequential order: the memory operations from all processors maintain a single sequential order

[Lamport’79]
Sequential Consistency (SC)

Per-processor program order: memory operations from individual processors maintain program order

Single sequential order: the memory operations from all processors maintain a single sequential order

[Lamport’79]
Sequential Consistency (SC)

Per-processor program order: memory operations from individual processors maintain program order

Single sequential order: the memory operations from all processors maintain a single sequential order

[λamport’79]
Problems with SC Enforcement
Problems with SC Enforcement

• Low Performance
Problems with SC Enforcement

• Low Performance
  • restrictions on performance-enhancing reordering of memory operations
Problems with SC Enforcement

• Low Performance
  • restrictions on performance-enhancing reordering of memory operations

• Or Complex Implementation

[Gharachorloo’91]
[Ranganathan’96]
[Gniady’97, SC++]

Ceze, Tuck, Montesinos, Torrellas
BulkSC: Bulk Enforcement of Sequential Consistency
Problems with SC Enforcement

• Low Performance
  • restrictions on performance-enhancing reordering of memory operations

• Or Complex Implementation
  • buffer long history of speculative memory accesses

[Gharachorloo’91]
[Ranganathan’96]
[Gniady’97, SC++]
Problems with SC Enforcement

• Low Performance
  • restrictions on performance-enhancing reordering of memory operations

• Or Complex Implementation
  • buffer long history of speculative memory accesses
  • check this history against coherence events and cache displacements

- [Gharachorloo ’91]
- [Ranganathan ’96]
- [Gniady ’97, SC++]
Problems with SC Enforcement

- **Low Performance**
  - restrictions on performance-enhancing reordering of memory operations

- **Or Complex Implementation**
  - buffer long history of speculative memory accesses
  - check this history against coherence events and cache displacements
  - coupled with key structures (LSQ, ROB, reg file, $)

---

Ceze, Tuck, Montesinos, Torrellas

BulkSC: Bulk Enforcement of Sequential Consistency

[Gharachorloo’91]
[Ranganathan’96]
[Gniady’97, SC++]
Problems with SC Enforcement

• Low Performance
  • restrictions on performance-enhancing reordering of memory operations

• Or Complex Implementation
  • buffer long history of speculative memory accesses
  • check this history against coherence events and cache displacements
  • coupled with key structures (LSQ, ROB, reg file, $)
  • typically fine-grain (instruction-level) undo

[Gharachorloo’91]
[Ranganathan’96]
[Gniady’97, SC++]
Problems with SC Enforcement

- Low Performance
  - restrictions on performance-enhancing reordering of memory operations

- Or Complex Implementation
  - buffer long history of speculative memory accesses
  - check this history against coherence events and cache displacements
  - coupled with key structures (LSQ, ROB, reg file, $)
  - typically fine-grain (instruction-level) undo

- Most current systems do not support SC

[Gharachorloo’91]
[Ranganathan’96]
[Gniady’97, SC++]
Problems with SC Enforcement

• Low Performance
  • restrictions on performance-enhancing reordering of memory operations

• Or Complex Implementation
  ➜ We would like to change that!
   • coupled with key structures (LSQ, ROB, reg file, $)
   • typically fine-grain (instruction-level) undo

• Most current systems do not support SC

[GHARACHORLOO’91]
[RANGANATHAN’96]
[GNIADY’97, SC++]
Problems with SC Enforcement

• Low Performance
  • restrictions on performance-enhancing reordering of memory operations

• Or Complex Implementation

→ We would like to change that!

→ Support SC with simple hardware and high performance
  • coupled with key structures (LSQ, ROB, reg file, $)
  • typically fine-grain (instruction-level) undo

• Most current systems do not support SC
BulkSC: Bulk Enforcement of Sequential Consistency
BulkSC: Bulk Enforcement of SC
BulkSC: Bulk Enforcement of SC
BulkSC: Bulk Enforcement of SC

Possible Global Order

P1: st A, ld C, st C, st D, st X
P2: st A, ld C, st C, st D, st X

Memory

Ceze, Tuck, Montesinos, Torrellas
BulkSC: Bulk Enforcement of SC

Possible Global Order

P1

st A
ld C
st C
st D
st X

P2

st A
ld C
st C
st D
ld D
st D
st X

P1

st A
ld C
st A
st C
st C
st C
st C

P2

st A
ld C
st A
st C
st C
st C
st C

P3

ld D
st D
ld D
st X

Memory

Ceze, Tuck, Montesinos, Torrellas

BulkSC: Bulk Enforcement of Sequential Consistency
BulkSC: Bulk Enforcement of SC

Possible Global Order:

- P1: st A, ld C, st C, st D, st X
- P2: st A, ld C, st C, st D, st X
- Memory:

Diagram:

- P1
- P2
- P3...
- PN

Ceze, Tuck, Montesinos, Torrellas
BulkSC: Bulk Enforcement of SC

- Group instructions into Chunks, enforce SC only at Chunk granularity

Possible Global Order:

<table>
<thead>
<tr>
<th>P1</th>
<th>P2</th>
</tr>
</thead>
<tbody>
<tr>
<td>st A</td>
<td>st A</td>
</tr>
<tr>
<td>ld C</td>
<td>st C</td>
</tr>
<tr>
<td>st C</td>
<td>ld D</td>
</tr>
<tr>
<td>st D</td>
<td>st X</td>
</tr>
</tbody>
</table>

Memory

Ceze, Tuck, Montesinos, Torrellas
BulkSC: Bulk Enforcement of SC

Group instructions into Chunks, enforce SC only at Chunk granularity

1. substantially reduce hardware complexity
Group instructions into Chunks, enforce SC only at Chunk granularity

1. substantially reduce hardware complexity
2. enable high performance
BulkSC: Bulk Enforcement of SC

Group instructions into Chunks, enforce SC only at Chunk granularity

1. substantially **reduce** hardware complexity
2. enable **high** performance
3. retain **programmability**
BulkSC: Bulk Enforcement of SC

- Group instructions into Chunks, enforce SC only at Chunk granularity

1. substantially reduce hardware complexity
2. enable high performance
3. retain programmability

- Execute a chunk atomically and in isolation, like a single instruction
Chunk Execution: Atomicity and Isolation
Chunk Execution: Atomicity and Isolation

P1

\[
\begin{align*}
st\ X \\
st\ Y
\end{align*}
\]

P2

\[
\begin{align*}
ld\ T \\
ld\ Z \\
st\ W
\end{align*}
\]
Chunk Execution: Atomicity and Isolation

Speculative Execution

P1

\[
\text{st X} \\
\text{st Y}
\]

P2

\[
\text{ld T} \\
\text{ld Z} \\
\text{st W}
\]
Chunk Execution: Atomicity and Isolation

**Atomicity**: all updates in the chunk are made visible to other processors at once (all or nothing)
Chunk Execution: Atomicity and Isolation

**Atomicity**: all updates in the chunk are made visible to other processors at once (all or nothing)
Chunk Execution: Atomicity and Isolation

**Speculative Execution**

**Atomicity**: all updates in the chunk are made visible to other processors at once (all or nothing)
Chunk Execution: Atomicity and Isolation

**Atomicity:** all updates in the chunk are made visible to other processors at once (all or nothing)

**Isolation:** a chunk should not see “outside” state changing during its execution
Chunk Execution: Atomicity and Isolation

**Atomicity**: all updates in the chunk are made visible to other processors at once (all or nothing)

**Isolation**: a chunk should not see “outside” state changing during its execution
Chunk Execution: Atomicity and Isolation

**Atomicity**: all updates in the chunk are made visible to other processors at once (all or nothing)

**Isolation**: a chunk should not see “outside” state changing during its execution
**Chunk Execution: Atomicity and Isolation**

**Atomicity**: all updates in the chunk are made visible to other processors at once (all or nothing)

**Isolation**: a chunk should not see “outside” state changing during its execution
Chunk Execution: Atomicity and Isolation

**Atomicity:** all updates in the chunk are made visible to other processors at once (all or nothing)

**Isolation:** a chunk should not see “outside” state changing during its execution
Requirements for Bulk Enforcement of SC
Requirements for Bulk Enforcement of SC

Per-processor program order:
*chunks* from individual processors
maintain program order
Requirements for Bulk Enforcement of SC

Per-processor program order: chunks from individual processors maintain program order

1 2
Requirements for Bulk Enforcement of SC

Per-processor program order: *chunks* from individual processors maintain program order

- **Global Order**
  - 1
  - 2

- **P1**
  - 1
  - 2
Requirements for Bulk Enforcement of SC

Per-processor program order: *chunks* from individual processors maintain program order

Single sequential order: *chunks* from all processors maintain a single sequential order
Requirements for Bulk Enforcement of SC

Per-processor program order: *chunks* from individual processors maintain program order

Single sequential order: *chunks* from all processors maintain a single sequential order
Requirements for Bulk Enforcement of SC

Per-processor program order: *chunks* from individual processors maintain program order

Single sequential order: *chunks* from all processors maintain a single sequential order
Bulk Operation [ISCA’06]
Bulk Operation [ISCA’06]

Addresses [Bloom’70]

---

Ceze, Tuck, Montesinos, Torrellas

BulkSC: Bulk Enforcement of Sequential Consistency
Bulk Operation [ISCA’06]

Addresses

[Bloom’70]
Bulk Operation [ISCA’06]

Signature Operations

Addresses

[Bloom’70]
Bulk Operation [ISCA’06]

Addresses

[Bloom’70]

Signature Operations

Bulk Framework

Ceze, Tuck, Montesinos, Torrellas

BulkSC: Bulk Enforcement of Sequential Consistency
Bulk Operation [ISCA’06]

Addresses

[Bloom’70]

H

S

Signature Operations
Bulk Framework

TM TLS

S1 ∩ S2

BulkSC: Bulk Enforcement of Sequential Consistency
Bulk Operation [ISCA’06]

Addresses

[Bloom’70]

Signatures Operations

Bulk Framework

Signature Operations
Bulk Operation [ISCA’06]

Addresses

[Bloom’70]

Signature Operations

Bulk Framework

TM TLS SC

BulkSC: Bulk Enforcement of Sequential Consistency
Efficiently Operating with Chunks
Efficiently Operating with Chunks

- Key idea: Summarize in hardware addresses accessed by a chunk into a pair of signatures
Efficiently Operating with Chunks

• Key idea: Summarize in hardware addresses accessed by a chunk into a pair of signatures
  
  Read R
  Write W
Efficiently Operating with Chunks

• Key idea: Summarize in hardware addresses accessed by a chunk into a pair of signatures
  Read R
  Write W

• Support signature operations with simple hardware
  • intersection, union, is-empty?, ...
  • much of the system operation is based on signatures
Efficiently Operating with Chunks

• Key idea: Summarize in hardware addresses accessed by a chunk into a pair of signatures

  Read \( R \)
  Write \( W \)

• Support signature operations with simple hardware
  • intersection, union, is-empty?, ...
  • much of the system operation is based on signatures

• At chunk commit
  • \( W \) signature is sent to other processors for disambiguation
  • \( R, W \) signatures are cleared
Chunk Atomicity and Isolation in Bulk

P1
Signatures  Cache
R1  
W1

P2
Signatures  Cache
R2  
W2
Chunk Atomicity and Isolation in Bulk

Ceze, Tuck, Montesinos, Torrellas

BulkSC: Bulk Enforcement of Sequential Consistency
Chunk Atomicity and Isolation in Bulk

P1

Signatures          Cache

R₁                      A
W₁

P2

Signatures          Cache

R₂                      
W₂

ld A
Chunk Atomicity and Isolation in Bulk

P1

Signatures
R₁
A
W₁

Cache

P2

Signatures
R₂
W₂

Cache

ld A
st B
Chunk Atomicity and Isolation in Bulk

Ceze, Tuck, Montesinos, Torrellas

BulkSC: Bulk Enforcement of Sequential Consistency
Chunk Atomicity and Isolation in Bulk

P1

Signatures
R1
W1

Cache
A
B

P2

Signatures
R2
W2

Cache

ld A
st B
st C
Chunk Atomicity and Isolation in Bulk

P1

Signatures
R1
W1

Cache
A
B
C

ld A
st B
st C

P2

Signatures
R2
W2

Cache

BulkSC: Bulk Enforcement of Sequential Consistency
Chunk Atomicity and Isolation in Bulk

P1

Signatures
R1
W1

Cache
A
B
C

ld A
st B
st C
ld D

P2

Signatures
R2
W2

Cache
Chunk Atomicity and Isolation in Bulk

P1

Signatures: R1, W1

Cache:

<table>
<thead>
<tr>
<th>A</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>C</td>
</tr>
</tbody>
</table>

P2

Signatures: R2, W2

Cache:

ld A
st B
st C
ld D
Chunk Atomicity and Isolation in Bulk

Ceze, Tuck, Montesinos, Torrellas

BulkSC: Bulk Enforcement of Sequential Consistency
Chunk Atomicity and Isolation in Bulk

BulkSC: Bulk Enforcement of Sequential Consistency
Chunk Atomicity and Isolation in Bulk

Ceze, Tuck, Montesinos, Torrellas

BulkSC: Bulk Enforcement of Sequential Consistency
Chunk Atomicity and Isolation in Bulk

P1

Signatures
R₁
A D
B C
W₁

Cache

ld A
st B
st C
ld D

commit
W₁

P2

Signatures
R₂
B C
W₂
E

Cache

ld B
st E
ld...

Ceze, Tuck, Montesinos, Torrellas
BulkSC: Bulk Enforcement of Sequential Consistency
Chunk Atomicity and Isolation in Bulk

P1

Signatures | Cache
---|---
R₁ | B
W₁ | C

commit

ld A
st B
st C
ld D

P2

Signatures | Cache
---|---
R₂ | B C
W₂ | E

ld B
st E
ld C
Chunk Atomicity and Isolation in Bulk

BulkSC: Bulk Enforcement of Sequential Consistency

Ceze, Tuck, Montesinos, Torrellas
Chunk Atomicity and Isolation in Bulk

P1

Signatures
R1
W1

Cache
B
C

P2

Signatures
R2
W2

Cache
B C
E

ld A
st B
st C
ld D

commit

W1

W1 \cap (W2 \cup R2) \neq \emptyset? \text{ (True)}
squash!

Ceze, Tuck, Montesinos, Torrellas
Chunk Atomicity and Isolation in Bulk

P1

Signatures
R1
W1

Cache
B
C

P2

Signatures
R2
W2

Cache
B
C
E

commit

W1 ≠ (W2 ∪ R2) ≠ ∅? (True)
squash!
Program Chunk Order
Program Chunk Order

• Chunks are **arbitrarily** defined by the hardware
Program Chunk Order

• Chunks are arbitrarily defined by the hardware
  • e.g. every 2000 dynamic instructions
Program Chunk Order

- Chunks are arbitrarily defined by the hardware
  - e.g. every 2000 dynamic instructions
- A processor commits chunks in *program order*
Program Chunk Order

- Chunks are arbitrarily defined by the hardware
  - e.g. every 2000 dynamic instructions
- A processor commits chunks in *program order*
- High-performance:
Program Chunk Order

• Chunks are arbitrarily defined by the hardware
  • e.g. every 2000 dynamic instructions

• A processor commits chunks in program order

• High-performance:
  • Instructions inside a chunk arbitrarily reordered
Program Chunk Order

• Chunks are arbitrarily defined by the hardware
  • e.g. every 2000 dynamic instructions

• A processor commits chunks in program order

• High-performance:
  • Instructions inside a chunk arbitrarily reordered

\[
\begin{array}{c}
\text{ld A} \\
\text{ld B} \\
\text{st C} \\
\text{st D}
\end{array}
\]
Program Chunk Order

• Chunks are **arbitrarily** defined by the hardware
  • e.g. every 2000 dynamic instructions

• A processor commits chunks in *program order*

• High-performance:
  • Instructions inside a chunk arbitrarily reordered

```
ld A  st D
ld B  st C
st C  ld B
st D  ld A
```
Program Chunk Order

- Chunks are arbitrarily defined by the hardware
  - e.g. every 2000 dynamic instructions

- A processor commits chunks in *program order*

- High-performance:
  - Instructions inside a chunk arbitrarily reordered
  
  - A processor can interleave the execution of instructions from 2 in-flight chunks

Ceze, Tuck, Montesinos, Torrellas

BulkSC: Bulk Enforcement of Sequential Consistency
Program Chunk Order

• Chunks are arbitrarily defined by the hardware
  • e.g. every 2000 dynamic instructions

• A processor commits chunks in program order

• High-performance:
  • Instructions inside a chunk arbitrarily reordered

• A processor can interleave the execution of instructions
  from 2 in-flight chunks
Program Chunk Order

- Chunks are arbitrarily defined by the hardware
  - e.g. every 2000 dynamic instructions
- A processor commits chunks in program order
- High-performance:
  - Instructions inside a chunk arbitrarily reordered
- A processor can interleave the execution of instructions from 2 in-flight chunks
Single Sequential Order
Single Sequential Order
Single Sequential Order

- Need an Arbiter for chunk commits
Single Sequential Order

• Need an Arbiter for chunk commits
• Naive:
  • allow a single commit at a time
Single Sequential Order

• Need an Arbiter for chunk commits
• Naive:
  • allow a single commit at a time
Allowing Simultaneous Chunk Commits
Allowing Simultaneous Chunk Commits
Allowing Simultaneous Chunk Commits

\[ \text{time} ~ P_1 ~ P_2 ~ P_3 \]

\[ \text{time} ~ P_1 ~ P_2 ~ P_3 \]
Allowing Simultaneous Chunk Commits

• Conditions:
Allowing Simultaneous Chunk Commits

- Conditions:
  - committing chunks’ memory operations should not intersect
Allowing Simultaneous Chunk Commits

• Conditions:
  • committing chunks’ memory operations should not intersect
    • both reads and writes
Allowing Simultaneous Chunk Commits

• Conditions:
  • committing chunks’ memory operations should not intersect
    • both reads and writes
  • this can be efficiently enforced with chunk signatures
Arbitrating Simultaneous Chunk Commits
Arbitrating Simultaneous Chunk Commits
Arbitrating Simultaneous Chunk Commits

P1

P2

P3

Arbiter

In-progress Commits
Arbitrating Simultaneous Chunk Commits

*Ceze, Tuck, Montesinos, Torrellas*

**BulkSC: Bulk Enforcement of Sequential Consistency**

Arbitrating Simultaneous Chunk Commits

<table>
<thead>
<tr>
<th>time</th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>C1</td>
<td>C2</td>
<td>C3</td>
</tr>
</tbody>
</table>

P1, P2, P3

In-progress Commits

Arbiter
Arbitrating Simultaneous Chunk Commits

(P1, R1) ?

In-progress Commits

Arbiter

P1

P2

P3

C1

C2

C3
Arbitrating Simultaneous Chunk Commits

(time P1, P2, P3)

\[ (W_1, R_1) ? \]

Ok

In-progress Commits

Arbiter

P1, P2, P3

C1, C2, C3
Arbitrating Simultaneous Chunk Commits

time P1  P2  P3

C1  C2  C3

P1 (W1, R1) ?

Ok

Arbiter

In-progress Commits

W1

P2

P3
Arbitrating Simultaneous Chunk Commits

\[
\text{time P1} \quad \text{P2} \quad \text{P3}
\]

(P1) \(W_1, R_1\)?

Ok In-progress Commits

Arbiter

W_1

Ceze, Tuck, Montesinos, Torrellas

BulkSC: Bulk Enforcement of Sequential Consistency
Arbitrating Simultaneous Chunk Commits

(Arbitration Example)

\[(W_1, R_1) \rightarrow \text{Ok} \rightarrow \text{In-progress Commits} \rightarrow \text{Arbiter} \rightarrow W_1 \]

\[(W_2, R_2) \]
Arbitrating Simultaneous Chunk Commits

W₁ ∩ (R₂ U W₂) = ∅?

Ceze, Tuck, Montesinos, Torrellas

BulkSC: Bulk Enforcement of Sequential Consistency
Arbitrating Simultaneous Chunk Commits

\[ \text{time P1, P2, P3} \]

\[ \text{P1: (W_1, R_1) ?} \]
\[ \text{Ok} \]

\[ \text{P2: (W_2, R_2) ?} \]

\[ \text{W_1 \cap (R_2 \cup W_2) = \emptyset? (True)} \]
Arbitrating Simultaneous Chunk Commits

\[ W_1 \cap (R_2 \cup W_2) = \emptyset? \ \text{(True)} \]
Arbitrating Simultaneous Chunk Commits

P1 P2 P3

\( (W_1, R_1) \) ? \( W_1 \cap (R_2 \cup W_2) = \emptyset \) (True)

W1
W2

P1
P2
P3
Arbitrating Simultaneous Chunk Commits

P1 P2 P3
\(C_1\) \(C_2\) \(C_3\)

Arbiter
In-progress Commits
\(W_1\ \cap \ (R_2 \cup W_2) = \emptyset\)? (True)

\begin{align*}
(W_1, R_1) &? \\
\text{Ok} \\
(W_2, R_2) &? \\
\text{Ok} \\
\end{align*}
Arbitrating Simultaneous Chunk Commits

\[
\begin{align*}
(W_1, R_1) & \to \text{Ok} \\
(W_2, R_2) & \to \text{Ok} \\
(W_3, R_3) & \to \text{Ok} \\
W_1 \cap (R_2 \cup W_2) & = \emptyset \quad \text{(True)}
\end{align*}
\]
Arbitrating Simultaneous Chunk Commits

(W₁, R₁) ?
(P1)
(W₂, R₂) ?
(P2)
(W₃, R₃) ?
(P3)

Arbiter
In-progress Commits
W₁
W₂

W₁ ∩ (R₂ ∪ W₂) = ∅? (True)
W₁ ∩ (R₃ ∪ W₃) = ∅?
Arbitrating Simultaneous Chunk Commits

\[
(W_1, R_1) ? \rightarrow \text{Ok} \\
(W_2, R_2) ? \rightarrow \text{Ok} \\
(W_3, R_3) ? \\
\]

\[
W_1 \cap (R_2 \cup W_2) = \emptyset? \quad \text{(True)} \\
W_1 \cap (R_3 \cup W_3) = \emptyset? \quad \text{(True)}
\]
Arbitrating Simultaneous Chunk Commits

(W₁, R₁) ?
Ok

(W₂, R₂) ?
Ok

(W₃, R₃) ?

P₁

P₂

P₃

Arbiter

In-progress
Commits

W₁ ∩ (R₂ ∪ W₂) = ∅? (True)
W₁ ∩ (R₃ ∪ W₃) = ∅? (True)
W₂ ∩ (R₃ ∪ W₃) = ∅?
Arbitrating Simultaneous Chunk Commits

\[ \text{time P1} \quad \text{P2} \quad \text{P3} \]

- \text{P1: } C1 \quad C2 \quad C3
- \text{P2: } (W_2, R_2) \quad \text{Ok}
- \text{P3: } (W_3, R_3) \quad \text{Ok}

Arbiter

In-progress Commits

- \text{W} \_1 \cap (R_2 \cup W_2) = \emptyset \? (True)
- \text{W} \_1 \cap (R_3 \cup W_3) = \emptyset \? (True)
- \text{W} \_2 \cap (R_3 \cup W_3) = \emptyset \? (False)
Arbitrating Simultaneous Chunk Commits

W₁ ∩ (R₂ ∪ W₂) = ∅? (True)
W₁ ∩ (R₃ ∪ W₃) = ∅? (True)
W₂ ∩ (R₃ ∪ W₃) = ∅? (False)
Arbitrating Simultaneous Chunk Commits

Ceze, Tuck, Montesinos, Torrellas

BulkSC: Bulk Enforcement of Sequential Consistency

W₁ ∩ (R₂ ∪ W₂) = ∅? (True)
W₁ ∩ (R₃ ∪ W₃) = ∅? (True)
W₂ ∩ (R₃ ∪ W₃) = ∅? (False)
Arbitrating Simultaneous Chunk Commits

Write signatures stay in the arbiter until commit completes
Complete Commit Process
Complete Commit Process
Complete Commit Process

P1 \(\rightarrow\) Arbiter \(\rightarrow\) Directory

\((W_1, R_1)\) ?

P2

P3
Complete Commit Process

\[ P1 \xrightarrow{(W_1, R_1)} \text{Arbiter} \xleftarrow{\text{Ok}} \]

Directory

\[ P2 \]

\[ P3 \]
Complete Commit Process

Directory

(W₁, R₁) ?
Ok

Arbiter

W₁

P1

P2

P3

Ceze, Tuck, Montesinos, Torrellas

BulkSC: Bulk Enforcement of Sequential Consistency
Complete Commit Process

P1 \( (W_1, R_1) \) ? \( \text{Ok} \)

Arbiter

W_1

Directory

P2

P3
Complete Commit Process

P1
(W1, R1) ?
Ok
Arbiter
W1
Directory
W1
Signature Expansion
P2
P3
Complete Commit Process

P1 \(\rightarrow\) Arbiter \(\rightarrow\) Directory

- \((W_1, R_1)\)?
- Ok
- W1
- W1

- Make P1 owner
- Determine sharers

P2 \(\rightarrow\) P3
Complete Commit Process

(P1, R1) ?

Ok

Arbiter

W1

Directory

Signature Expansion

- Make P1 owner
- Determine sharers

P2

P3

W1

W1

W1

P1

Ceze, Tuck, Montesinos, Torrellas  BulkSC: Bulk Enforcement of Sequential Consistency
Complete Commit Process

P1 \rightarrow \text{Arbiter} \rightarrow \text{Directory} \rightarrow \text{Disambiguate}

(P1, (W_1, R_1)) \rightarrow \text{Ok} \rightarrow W_1 \rightarrow \text{Signature Expansion} \rightarrow \text{W}_1 \rightarrow \text{P2, P3}

- Make P1 owner
- Determine sharers
Complete Commit Process

- P1
- Arbiter
  - W1
- Directory
  - Signature Expansion
    - Make P1 owner
    - Determine sharers
- P2
- P3
- (W1, R1) ?
- Ok
- W1
- Disambiguate
- W1
- Ack
- W1
Complete Commit Process

![Diagram of Complete Commit Process]

- **P1** (Writer 1, Reader 1) interacts with the **Arbiter**.
- **Arbiter** decides if P1 can proceed.
- If approved, P1 proceeds to the **Directory**.
- The **Directory** performs signature expansion and decides on ownership and sharers.
- **Disambiguate** is the final step to resolve any conflicts.

Steps:

- Make P1 owner
- Determine sharers
Complete Commit Process

- P1
  - (W₁, R₁) ?
  - Ok

Arbiter

- W₁
- Ack W₁

Directory

- Signature Expansion
- Make P1 owner
- Determine sharers

- W₁
- Ack

- W₁
- Ack

- Disambiguate

P2

P3
Complete Commit Process

P1 \( (W_1, R_1) \) \( \xrightarrow{\text{Ok}} \) Arbiter \( \xrightarrow{W_1} \) Directory

- Signature Expansion
- Make P1 owner
- Determine sharers

\( \xrightarrow{\text{Ack} W_1} \) P1 owner

\( \xrightarrow{\text{Ack} W_1} \) P2

\( \xrightarrow{\text{Ack} W_1} \) P3

\( \xrightarrow{\text{Ack}} \) P2

\( \xrightarrow{\text{Ack}} \) P3

Disambiguate
Complete Commit Process

(W₁, R₁) ?

Arbiter

(W₁)

Directory

Signature Expansion

• Make P1 owner
• Determine sharers

P1

P2

P3

Ack W₁

Ok

W₁

Ack

Disambiguate

Ack

Ack
Complete Commit Process

The whole process only uses signatures
Complete Commit Process

The whole process only uses signatures
Can support a distributed arbiter
Advantages of BulkSC
Advantages of BulkSC

• **Simplifies the HW complexity** of high-performance SC
Advantages of BulkSC

- Simplifies the HW complexity of high-performance SC
  - decouples consistency enforcement from core micro-architecture and caches
Advantages of BulkSC

• **Simplifies the HW complexity** of high-performance SC

  • decouples consistency enforcement from core micro-architecture and caches
Advantages of BulkSC

• **Simplifies the HW complexity of high-performance SC**
  • decouples consistency enforcement from core micro-architecture and caches
  • single-thread optimizations without worrying about multiprocessor issues
Advantages of BulkSC

• **Simplifies the HW complexity** of high-performance SC
  - decouples consistency enforcement from core micro-architecture and caches
  - single-thread optimizations without worrying about multiprocessor issues
  - no associative structures in the processor
Advantages of BulkSC

- **Simplifies the HW complexity of high-performance SC**
  - decouples consistency enforcement from core micro-architecture and caches
  - single-thread optimizations without worrying about multiprocessor issues
  - no associative structures in the processor
  - no extra bits in the cache for versioning [ISCA’06]

---

![Diagram showing the differences between Traditional Processor and BulkSC Processor]
Advantages of BulkSC

- Simplifies the HW complexity of high-performance SC
  - decouples consistency enforcement from core micro-architecture and caches
  - single-thread optimizations without worrying about multiprocessor issues
  - no associative structures in the processor
  - no extra bits in the cache for versioning [ISCA’06]

- Memory ordering framework for MPs
Advantages of BulkSC

- **Simplifies the HW complexity of high-performance SC**
  - decouples consistency enforcement from core micro-architecture and caches
  - single-thread optimizations without worrying about multiprocessor issues
  - no associative structures in the processor
  - no extra bits in the cache for versioning [ISCA’06]

- **Memory ordering framework for MPs**
  - small extension to support TM, TLS, ...
Summary of Evaluation
Summary of Evaluation

• Cycle-accurate simulations [SESC]
Summary of Evaluation

• Cycle-accurate simulations [SESC]

![Graph showing evaluation results for SPLASH2-GM, sjbb2000, and sweb2005]
Summary of Evaluation

- **Cycle-accurate simulations [SESC]**

  - **Result**: SC with performance comparable to RC (1%) with little BW cost (5-13%)
Summary of Evaluation

•**Cycle-accurate simulations [SESC]**

• **Result:** SC with performance comparable to RC (1%) with little BW cost (5-13%)

• **Much simpler hardware than SC++**
Also on the paper...
Also on the paper...

• Interaction with explicit synchronization, TM
Also on the paper...

- Interaction with explicit synchronization, TM
- Forward progress
Also on the paper...

- Interaction with explicit synchronization, TM
- Forward progress
- I/O
Also on the paper...

• Interaction with explicit synchronization, TM
• Forward progress
• I/O
• Distributed arbiter
Also on the paper...

- Interaction with explicit synchronization, TM
- Forward progress
- I/O
- Distributed arbiter
- Directory design for signatures
Also on the paper...

• Interaction with explicit synchronization, TM
• Forward progress
• I/O
• Distributed arbiter
• Directory design for signatures
• Two optimizations for private data
Also on the paper...

• Interaction with explicit synchronization, TM
• Forward progress
• I/O
• Distributed arbiter
• Directory design for signatures
• Two optimizations for private data
• Discussion on scalability
Conclusions
Conclusions

- Presented BulkSC
Conclusions

• Presented BulkSC
  • coarse-grain, signature-based enforcement of SC
Conclusions

• Presented BulkSC
  • coarse-grain, signature-based enforcement of SC

• Performance comparable to RC
Conclusions

• Presented BulkSC
  • coarse-grain, signature-based enforcement of SC

• Performance comparable to RC

• Simplicity: decouple consistency enforcement from processor design
Conclusions

• Presented BulkSC
  • coarse-grain, signature-based enforcement of SC

• Performance comparable to RC

• Simplicity: decouple consistency enforcement from processor design

• Independent of network ordering properties
Conclusions

• Presented BulkSC
  • coarse-grain, signature-based enforcement of SC

• Performance comparable to RC

• Simplicity: decouple consistency enforcement from processor design

• Independent of network ordering properties

• Generic ordering framework for speculative multiprocessors
BulkSC: Bulk Enforcement of Sequential Consistency

Luis Ceze, James Tuck, Pablo Montesinos, Josep Torrellas

University of Illinois at Urbana-Champaign

http://iacoma.cs.uiuc.edu/

ISCA 2007, San Diego, CA