Speculative Versioning Cache: Unifying Speculation and Coherence

Sridhar Gopal
T.N. Vijaykumar†, Jim Smith, Guri Sohi

Multiscalar Project
Computer Sciences Department
University of Wisconsin, Madison

†Electrical and Computer Engineering, Purdue University
Motivation

Enable a single hardware platform to support ...

1. SMP execution model - explicitly parallel programs
   *Private* L1 coherent caches for high performance

2. Hierarchical execution model - sequential programs
   - Multiscalar, Trace Processors, Agassiz, Hydra, Stampede, WarpEngine
   Memory Renaming or Speculative Versioning required

**SVC = Cache Coherence + Speculative Versioning**
Outline

Motivation

Hierarchical Execution Model

- Hierarchical Execution
- Multiscalar

Speculative Versioning

Unifying Speculation and Coherence

Speculative Versioning Cache (SVC)

Address Resolution Buffer (ARB)

Performance Evaluation

Conclusions
Hierarchical Execution Model

**Extract Parallelism in Sequential Programs**

1. Using Speculation
2. Using Multiple Processors

- Group instructions into *tasks*, *traces* or *speculation regions*
- Employ task level speculation

**Sequential Execution**

- A, B: Tasks

**Hierarchical Execution**

- Predict
- Predict
- Predict

Speculatively execute A and B in parallel

Dependences between A and B?
Hierarchical Execution: Multiscalar

Higher Level: Tasks

- Incorrect control prediction: squash task state and re-execute
- Task completed: commit task state sequentially

Lower Level: Instructions

Register dependences: Hardware + Software

Memory dependences: Hierarchical hardware

- Intra-task: Load Store Queue in each processor
- Inter-task: SVC or ARB
Outline

Motivation

Hierarchical Execution Model

Speculative Versioning

- Problem: Guarantee sequential program semantics
- Example execution orders
- Key requirements of a solution

Unifying Speculation and Coherence

Speculative Versioning Cache (SVC)

Address Resolution Buffer (ARB)

Performance Evaluation

Conclusions
Speculative Versioning: Problem

DIFFERENT Address: Accesses can be reordered

SAME ADDRESS: WHICH ORDERS ARE ALLOWED?
Execution Order: Examples

Program Order: Ld/St to the same address

<table>
<thead>
<tr>
<th>Program Order</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>SL SL SL</td>
<td>T0</td>
</tr>
<tr>
<td></td>
<td>T1</td>
</tr>
<tr>
<td></td>
<td>T2</td>
</tr>
<tr>
<td></td>
<td>T3</td>
</tr>
<tr>
<td></td>
<td>⊣</td>
</tr>
</tbody>
</table>

Execution Orders

- ✔️ SL SL SL: No action!
- ✔️ S S L L: Out of order loads okay!
- ✘ S L L S: Blue load is incorrect!
- ✗ S S L L: Stores out of order, Write back in order

CANNOT REORDER DEPENDENT LOADS AND STORES
Speculative Versioning: Solution

Definition: Version of a location

- Each store creates a new version

What should a solution provide?

- Buffer multiple versions and track order
- Supply the correct version for a load
- Detect incorrect loads
- Write back versions in order

Definition: Version Ordering List (VOL) of a location

- Execution order of loads and stores

DIRECTORY OF VERSION ORDERING LISTS
Unification

Speculative Versioning

- Multiple copies of multiple speculative versions
- Execution order tracked using an ordered list
  Multiple Reader Multiple Writer Protocol

Cache Coherence

- Multiple copies of a single version
- Sharers tracked using an unordered list
  Multiple Reader Single Writer Protocol

$\text{SVC} = \text{Cache Coherence} + \text{Speculative Versioning}$
Outline

Motivation
Hierarchical Execution Model
Speculative Versioning
Speculative Versioning Cache (SVC)
  • Simple Cache Coherence Protocol
  • Base SVC Protocol
  • Advanced SVC Protocol
  • Examples
Address Resolution Buffer (ARB)
Performance Evaluation
Conclusions
Speculative Versioning Cache

Extensions to SMP coherent cache

- Both speculative and committed data
- More state information
- VOL: a pointer in each line
- VCL: maintain VOL on bus requests
TALK Cache block size = Load/Store granularity = One word

PAPER Realistic linesize, Cast-outs
Base SVC Design: Processor Requests

- **Load** bit or **DL**: detect incorrect loads
- DL remembers use before definition
- Write back **all** dirty lines on commit
- Invalidate **all** clean lines on squash
BusRead: Supplying the Correct Version

- VCL: Closest older version is **Flushed**
- Self Loops: Cannot write back speculative versions

**Examples**

Program Order

<table>
<thead>
<tr>
<th>T0</th>
<th>T1</th>
<th>T2</th>
<th>T3</th>
<th>T4</th>
</tr>
</thead>
<tbody>
<tr>
<td>D</td>
<td>D</td>
<td>C</td>
<td>I</td>
<td>D</td>
</tr>
<tr>
<td>C</td>
<td>I</td>
<td>C</td>
<td>I</td>
<td>D</td>
</tr>
</tbody>
</table>

Newer

BRd/Fl

Fl: Flush

Self Loops

BRd/Fl

C

D

I

DL
BusWrite: Detecting Incorrect Loads

- VCL: Copies used by newer tasks **Invalidated**
- DL is also invalidated because of incorrect use

Examples

Program Order:

```
T_0  T_1  T_2  T_3  T_4
```

1. C   I   C   C   D
2. C   I   C   C   DL
Base Design Problems and Solutions

Write back dirty lines on commit

- New tasks begin after all write backs
- **Bottleneck**: Tasks commit sequentially
  
  ➔ Commit (C) bit

Invalidate clean lines on commit/squash

- Every task begins with a **cold** cache
  
  ➔ sTale (T) and Architectural (A) bits

Reference Spreading

- Private caches: **Multiple** misses to the **same** data
  
  ➔ Snarfing
Outline

Motivation
Hierarchical Execution Model
Speculative Versioning
Unifying Speculation and Coherence
Speculative Versioning Cache (SVC)
Address Resolution Buffer (ARB)
  • Previously proposed shared cache solution
Performance Evaluation
Conclusions
Shared Cache Solution: ARB

- Shared speculative ARB + Shared Cache
  - Every load and store incurs interconnect latency
  - On commits, speculative versions **must** be written back
  - Allocates space for non-existent versions
Performance Evaluation: SPEC95

4-way Multiscalar  2-way ooo superscalar Ps
64KB data cache with 8KB ARB  4x16KB SVC

TRADES-OFF HIT-RATE FOR HIT-LATENCY
Speculative Versioning Cache

- Unifies speculative versioning and cache coherence
- Private cache solution for speculative versioning
- Hold both speculative and committed data in one cache
- Decouples write backs from commits
- Eliminates unnecessary write backs
- Trades-off hit-rate for hit-latency

*SVC = Cache Coherence + Speculative Versioning*
Decoupling Commit from Write backs

Explicit pointers necessary to track order among versions
VCL determines the CD line to be written back;
all other CD lines are purged
Keeping Cache Warm: After Commits

Problem
- All C lines are invalidated when a task commits

Solution
- sTale bit: to distinguish between sTale and clean copies

CC LINES ARE CACHE HITS ON LOADS
Keeping Cache Warm: After Squashes

**Problem**
- A squashed task could fetch the same data multiple times

**Solution**
- Architectural bit: to retain architectural versions after a squash

**AC LINES ARE CACHE HITS ON LOADS**
Base SVC Design: Examples

<table>
<thead>
<tr>
<th>Task #</th>
<th>Program Order</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>st 0, A</td>
</tr>
<tr>
<td>1</td>
<td>st 1, A</td>
</tr>
<tr>
<td>2</td>
<td>Id r, A</td>
</tr>
<tr>
<td>3</td>
<td>st 3, A</td>
</tr>
<tr>
<td>5</td>
<td>st 5, A</td>
</tr>
<tr>
<td>6</td>
<td>Id r, A</td>
</tr>
</tbody>
</table>

Program order:
- 0: S
- 1: S
- 2: W/Z
- 3: L

Execution time order:
- 0: S
- 1: S
- 2: L
- 3: S

Valid state (V): S
Store state (S): W
Load state (L): L
Data states:
- S: Store
- L: Load

Tag | V | S | L | Data
---|---|---|---|---
0 | S | 0 |   |   
1 | S | 3 | W | Y 
2 |   |   | Z |   
3 |   |   |   | 1 
5 |   |   |   |   
6 |   |   | W | Y 

Note: The diagram illustrates the state transitions for each task and the corresponding data states.
Adv. SVC Design I: Efficient Commits

<table>
<thead>
<tr>
<th>Task #</th>
<th>V</th>
<th>S</th>
<th>L</th>
<th>C</th>
<th>Pointer</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>st 0, A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>st 1, A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>ld r, A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>st 3, A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>st 5, A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>ld r, A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Program Order:

- Task 0: st 0, A
- Task 1: st 1, A
- Task 2: ld r, A
- Task 3: st 3, A
- Task 4: st 5, A
- Task 5: ld r, A

Tag: V: Valid, S: Store, L: Load, C: Commit

Task 0:
- st 0, A
- X/4
- Y/5
- CS 0
- CS 1

Task 1:
- st 1, A
- X/4
- Y/5
- CS 1
- CS 1

Task 2:
- ld r, A
- X/4
- Y/5
- CS 1
- CS 1

Task 3:
- st 3, A
- X/4
- Y/5
- CS 1
- CS 1

Task 4:
- st 5, A
- X/4
- Y/5
- CS 1
- CS 1

Task 5:
- ld r, A
- X/4
- Y/5
- CS 1
- CS 1

Task 6:
- st 5, A
- X/4
- Y/5
- CS 1
- CS 1

© Sridhar Gopal, UW-Madison
Speculative Versioning Cache
February 3, 1998
Page 26
Multiple Committed Versions

CS 0

X/20

W/23

Y/21

Z/22

CS 3

Id r, A

CS 3

Id r, A

CS W 0

X/20

W/23

Y/21

Z/22

CS - 3

Id r, A

CS X 3

X/20

W/23

Y/21

Z/22

CS 4

X/20

W/23

Y/21

Z/22

CS 3

Id r, A

CS X 3

X/20

W/23

Y/21

Z/22

CS 4

X/20

W/23

Y/21

Z/22

CS 3

Id r, A
Adv. SVC Design II: Stale Copies

S Y 0
X/0
W/3 Y/1 S Z 1
L - 1

CS Y 0
X/4
W/7 Y/5 CS Z 1
CL - 1
Id r, A

S - 3 W/3 Y/5 CS Z 1
L W 1

CS - 3 W/7 Y/5 CS Z 1
CL W 1
Id r, A

X/2
Z/0
W/1
Z/2

Adv. SVC Design II: sTale bit

<table>
<thead>
<tr>
<th>Tag</th>
<th>V</th>
<th>S</th>
<th>L</th>
<th>C</th>
<th>T</th>
<th>A</th>
<th>Pointer</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

V: Valid  
S: Store  
L: Load  
C: Commit  
T: sTale  
A: Architectural

Tag | V | S | L | C | T | A | Pointer | Data
----|---|---|---|---|---|---|---------|------
ST | Y | 0 |   |   | X | 0 |         |      |
W | 3 | Y | 1 | S | Z | 1 |         |      |
L  | - | 1 |   |   |   |   |         |      |

ST | Y | 0 |
W | 3 |
S | Z | 1 |
L  | - | 1 |

CST | Y | 0 |
W | 7 |
CS | Z | 1 |
CL | - | 1 |

Id r, A

CST | Y | 0 |
W | 3 |
S | Z | 1 |
LT | W | 1 |

CST | Y | 0 |
W | 7 |
CS | Z | 1 |
CLT| W | 1 |

Id r, A
Adv. SVC Designs: Task Squashes
SVC Design: Realistic Linesize

False sharing

• Unnecessary invalidates similar to that for parallel programs
  • Worse, these invalidates lead to false squashes

Versioning Block

• Similar to Sub-blocking (Transfer and Address Blocks)
• Definition: Granularity at which speculative versioning is provided
• Replicate only the L and S bits for every versioning block