BulkSMT: Designing SMT Processors for Atomic-Block Execution

Xuehai Qian, Benjamin Sahelices and Josep Torrellas
University of Illinois

To appear at HPCA 2012

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

• Architectures that continuously execute **Atomic Blocks**
  • Performance and programmability advantages [Hammond 04], [Ahn 10]
  • All proposals use single context cores

• What if we used **Simultaneous Multithreading (SMT) cores**?
  • Enable a better utilization of the hardware
  • Fast/local communication between contexts
  • Enable **higher-concurrency** forms of atomic block execution
Contributions
Contributions

- **BulkSMT**: first SMT design that supports atomic-block (transactional) execution
  - Enables concurrency between dependent blocks
- Analysis of design space
  - SQUASH on conflict
  - STALL on conflict
  - ORDER on conflict
- Design of a multicore of BulkSMTs
- BulkSMT is cost Effective
  - Higher performance for the same core count
  - Competitive performance for 1/4 of the cores
Outline

- Motivation
- Designing BulkSMT
  - Design Space
- Hardware Mechanisms
- Multicore of BulkSMTs
- Evaluation
Designing SMT For Blocked Execution

Contexts

L1/L2

• Version Management:
• Conflict Detection:
• Conflict Resolution:
Designing SMT For Blocked Execution

• Version Management: Eager
• Conflict Detection:
• Conflict Resolution:

L1/L2

Contexts

No disp

spec wr
Designing SMT For Blocked Execution

- Version Management: Eager
- Conflict Detection: Eager
- Conflict Resolution:

L1/L2 spec rd wr

Contexts spec wr

No disp
Designing SMT For Blocked Execution

- Version Management: Eager
- Conflict Detection: Eager
- Conflict Resolution:
  - SQUASH
  - STALL
  - ORDER
SQUASH Design

BulkSMT: Designing SMT Processors for Atomic Block Execution

Xuehai Qian

Thursday, December 8, 2011
SQUASH Design

BulkSMT: Designing SMT Processors for Atomic Block Execution
SQUASH Design

BulkSMT: Designing SMT Processors for Atomic Block Execution
SQUASH Design

BulkSMT: Designing SMT Processors for Atomic Block Execution

Xuehai Qian

Thursday, December 8, 2011
STALL Design

BulkSMT: Designing SMT Processors for Atomic Block Execution
STALL Design

BulkSMT: Designing SMT Processors for Atomic Block Execution
STALL Design
STALL Design
STALL Design

BulkSMT: Designing SMT Processors for Atomic Block Execution
ORDER Design

BulkSMT: Designing SMT Processors for Atomic Block Execution
ORDER Design

P0

L1$

L2$

wr x

rd x

wr x

rd x

BulkSMT: Designing SMT Processors for Atomic Block Execution
ORDER Design

BulkSMT: Designing SMT Processors for Atomic Block Execution
ORDER Design

Xuehai Qian

BulkSMT: Designing SMT Processors for Atomic Block Execution
ORDER Design

BulkSMT: Designing SMT Processors for Atomic Block Execution
STALL Design Issues
STALL Design Issues

- On a dependence:
  - Stall the destination block
  - HW records source and destination blocks
- On commit/squash of source block
  - Wake up stalled destination block
STALL Design Issues
STALL Design Issues

- Transitive stall OK
- Block allowed to stall on an already stalled block
STALL Design Issues

- Transitive stall OK
- Block allowed to stall on an already stalled block
STALL Design Issues

- Transitive stall OK
- Block allowed to stall on an already stalled block
STALL Design Issues

- Transitive stall OK
- Block allowed to stall on an already stalled block
- Cycle not OK
  - When a stall cycle is formed, block that closes the cycle is squashed

T0

wr y

wr x

T1

rd x

T1 wakes up T0
T0 wakes up T2

T2

wr y

Total order of blocks:
T1 → T0 → T2
STALL Design Issues

• Transitive stall OK
• Block allowed to stall on an already stalled block
• Cycle not OK
• When a stall cycle is formed, block that closes the cycle is squashed

T0 ➔ T1 ➔ T2

Total order of blocks: T1 ➔ T0 ➔ T2

T0 wakes up T2
T0 wakes up T2

wr y
wr x
rd x

wr y
wr x
rd x

wr y
STALL Design Issues

- Transitive stall OK
- Block allowed to stall on an already stalled block
- Cycle not OK
  - When a stall cycle is formed, block that closes the cycle is squashed

\[
\begin{align*}
\text{T0} & \quad \text{T1} \quad \text{T2} \\
\text{wr y} & \quad \text{rd x} & \quad \text{wr y} \\
\text{wr x} & \quad & \\
\end{align*}
\]

T1 wakes up T0
T0 wakes up T2
Total order of blocks: T1 → T0 → T2

\[
\begin{align*}
\text{T0} & \quad \text{T1} \\
\text{wr y} & \quad \text{rd x} \\
\text{wr x} & \quad \text{wr y} \\
\end{align*}
\]

Squash T1
Total order of blocks: T0 → T1
ORDER Design Issues (I)
ORDER Design Issues (I)

- On a dependence:
  - HW records source and destination block; Both blocks proceed
  - HW will enforce the same order in commit
ORDER Design Issues (I)

- On a dependence:
  - HW records source and destination block; Both blocks proceed
  - HW will enforce the same order in commit
  - Transitive order is OK
ORDER Design Issues (I)

- On a dependence:
  - HW records source and destination block; Both blocks proceed
  - HW will enforce the same order in commit
  - Transitive order is OK

Total order of blocks:
T1 → T0 → T2
ORDER Design Issues (I)

- On a dependence:
  - HW records source and destination block; Both blocks proceed
  - HW will enforce the same order in commit
- Transitive order is OK
- Cycle is not OK
- On cycle, HW breaks it by squashing one or more blocks involved in the cycle

T0 T1 T2

wr y rd x wr x

Total order of blocks:
T1 → T0 → T2

wr y
ORDER Design Issues (I)

- On a dependence:
  - HW records source and destination block; Both blocks proceed
  - HW will enforce the same order in commit
- Transitive order is OK
- Cycle is not OK

  - On cycle, HW breaks it by squashing one or more blocks involved in the cycle

T0 T1 T2

wr y rd x
wr x rd x

Total order of blocks: T1 → T0 → T2
ORDER Design Issues (II)
ORDER Design Issues (II)

• On a cycle: different dependences have different squash requirement
ORDER Design Issues (II)

- On a cycle: different dependences have different squash requirement

RAW

Squash T0 $\Rightarrow$ Squash T1
Squash T1 $\not\Rightarrow$ Squash T0
ORDER Design Issues (II)

- On a cycle: different dependences have different squash requirement

RAW

\[
\begin{align*}
T0 & \quad T1 \\
wr & \quad rd
\end{align*}
\]

Squash T0 ⇒ Squash T1
Squash T1 ⇒ Squash T0

WAW

\[
\begin{align*}
T0 & \quad T1 \\
wr & \quad wr
\end{align*}
\]

Squash T0 ⇒ Squash T1
Squash T1 ⇒ Squash T0
ORDER Design Issues (II)

• On a cycle: different dependences have different squash requirement

RAW

\[
\begin{align*}
\text{Squash T0} &\Rightarrow \text{Squash T1} \\
\text{Squash T1} &\not\Rightarrow \text{Squash T0}
\end{align*}
\]

WAW

\[
\begin{align*}
\text{Squash T0} &\Rightarrow \text{Squash T1} \\
\text{Squash T1} &\Rightarrow \text{Squash T0}
\end{align*}
\]

WAR

\[
\begin{align*}
\text{Squash T0} &\not\Rightarrow \text{Squash T1} \\
\text{Squash T1} &\not\Rightarrow \text{Squash T0}
\end{align*}
\]
Outline

• Motivation
• Designing BulkSMT
  • Design Space
  • Hardware Mechanisms
• Multicore of BulkSMTs
• Evaluation
## Hardware Mechanisms

<table>
<thead>
<tr>
<th>Mechanism</th>
<th>Function</th>
<th>Impl.</th>
<th>Designs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access Recording and Conflict Detection</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cycle Detection</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Advanced Conflict Recording</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Squash Set Generation</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# Hardware Mechanisms

<table>
<thead>
<tr>
<th>Mechanism</th>
<th>Function</th>
<th>Impl.</th>
<th>Designs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access Recording and Conflict Detection</td>
<td>Record the addresses accessed by each thread and detect when two threads have a data conflict</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cycle Detection</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Advanced Conflict Recording</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Squash Set Generation</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Hardware Mechanisms

<table>
<thead>
<tr>
<th>Mechanism</th>
<th>Function</th>
<th>Impl.</th>
<th>Designs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access Recording and Conflict Detection</td>
<td>Record the addresses accessed by each thread and detect when two threads have a data conflict</td>
<td>Access Bits in cache and related logic</td>
<td></td>
</tr>
<tr>
<td>Cycle Detection</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Advanced Conflict Recording</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Squash Set Generation</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Xuehai Qian

**BulkSMT: Designing SMT Processors for Atomic Block Execution**
# Hardware Mechanisms

<table>
<thead>
<tr>
<th>Mechanism</th>
<th>Function</th>
<th>Impl.</th>
<th>Designs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access Recording and Conflict Detection</td>
<td>Record the addresses accessed by each thread and detect when two threads have a data conflict</td>
<td>Access Bits in cache and related logic</td>
<td>All</td>
</tr>
<tr>
<td>Cycle Detection</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Advanced Conflict Recording</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Squash Set Generation</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Access Recording

- LW: Last Writer context-ID
- R[i]: Read bit-mask
- Sp: Speculative bit

Cache (Shared by all contexts)
Access Recording

- LW: Last Writer context-ID
- R[i]: Read bit-mask
- Sp: Speculative bit
- Read from context $k$: set $R[k]$
Access Recording

- LW: Last Writer context-ID
- R[i]: Read bit-mask
- Sp: Speculative bit

- Read from context $k$: set $R[k]$
- Write from context $k$: $Sp \leftarrow 1$, $LW \leftarrow k$
Conflict Detection
Conflict Detection

- Read After Write (RAW)
  - Load from context $k$ reads cache line and $(Sp == 1) \&\& (LW != k)$
- Write After Read (WAR)
  - ...
- Write After Write (WAW)
  - ...

Xuehai Qian

BulkSMT: Designing SMT Processors for Atomic Block Execution
Commit and Squash
Commit and Squash

• On commit of context $k$:
  • Flash clear of the $R[k]$ bit of all the cache lines
  • Flash conditional-clear the $Sp$ bits of any line whose $(LW == k)$

• On Squash (context $k$):
  • ...

Xuehai Qian
BulkSMT: Designing SMT Processors for Atomic Block Execution
Hardware Mechanisms

<table>
<thead>
<tr>
<th>Mechanism</th>
<th>Function</th>
<th>Impl.</th>
<th>Designs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access Recording and Conflict Detection</td>
<td>Record the addresses accessed by each thread and detect when two threads have a data conflict</td>
<td>Access Bits in cache and related logic</td>
<td>All</td>
</tr>
<tr>
<td>Cycle Detection</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Advanced Conflict Recording</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Squash Set Generation</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Hardware Mechanisms

<table>
<thead>
<tr>
<th>Mechanism</th>
<th>Function</th>
<th>Impl.</th>
<th>Designs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access Recording and Conflict Detection</td>
<td>Record the addresses accessed by each thread and detect when two threads have a data conflict</td>
<td>Access Bits in cache and related logic</td>
<td>All</td>
</tr>
<tr>
<td>Cycle Detection</td>
<td>Record data conflicts and their ordering, and detect conflict cycles</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Advanced Conflict Recording</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Squash Set Generation</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# Hardware Mechanisms

<table>
<thead>
<tr>
<th>Mechanism</th>
<th>Function</th>
<th>Impl.</th>
<th>Designs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access Recording and Conflict Detection</td>
<td>Record the addresses accessed by each thread and detect when two threads have a data conflict</td>
<td>Access Bits in cache and related logic</td>
<td>All</td>
</tr>
<tr>
<td>Cycle Detection</td>
<td>Record data conflicts and their ordering, and detect conflict cycles</td>
<td>Dependence Table and Cycle Table</td>
<td></td>
</tr>
<tr>
<td>Advanced Conflict Recording</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Squash Set Generation</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Hardware Mechanisms

<table>
<thead>
<tr>
<th>Mechanism</th>
<th>Function</th>
<th>Impl.</th>
<th>Designs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access Recording and Conflict Detection</td>
<td>Record the addresses accessed by each thread and detect when two threads have a data conflict</td>
<td>Access Bits in cache and related logic</td>
<td>All</td>
</tr>
<tr>
<td>Cycle Detection</td>
<td>Record data conflicts and their ordering, and detect conflict cycles</td>
<td>Dependence Table and Cycle Table</td>
<td>STALL ORDER</td>
</tr>
<tr>
<td>Advanced Conflict Recording</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Squash Set Generation</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Cycle Detection: HW Structures
Cycle Detection: HW Structures

Dependence Table (DT)

Records ordered dependences

$Ti \rightarrow Tj$
Cycle Detection: HW Structures

Dependence Table (DT)
- Records ordered dependences
- \( Ti \rightarrow Tj \)

Cycle Table (CT)
- In background:
- Finds cycles of dependences
- Cycle = bit in diagonal

\( Ti \) \( Tj \)

\( n (=4) \)

\( i \)

\( j \)

\( n \)

\( i \)

\( j \)
Cycle Detection

T0  T1  T2

DT

CT
Cycle Detection
Cycle Detection

T0  T1  T2

DT

CT

d1

d1

BulkSMT: Designing SMT Processors for Atomic Block Execution
Cycle Detection

T0  T1  T2

DT

CT
d1

 BulkSMT: Designing SMT Processors for Atomic Block Execution
Cycle Detection

(d1)

(d2)

T0  T1  T2

DT

CT

d1
Cycle Detection

![Diagram showing cycle detection]

T0 → T1 → T2

DT → CT

d1 → d2

BulkSMT: Designing SMT Processors for Atomic Block Execution

Xuehai Qian

Thursday, December 8, 2011
Cycle Detection

T0  T1  T2

T0: d1
T1: d2
T2: d1

DT  DT

CT  CT

Xuehai Qian

BulkSMT: Designing SMT Processors for Atomic Block Execution

Thursday, December 8, 2011
Cycle Detection

T0 T1 T2

d1 d2

DT CT DT CT

1 1 1 1

d1 d2

Xuehai Qian

BulkSMT: Designing SMT Processors for Atomic Block Execution

Thursday, December 8, 2011
Cycle Detection

Propagation of $d_2$ dependence:

$Col_{dst} \neq Col_{src}$

$Row_{src} \neq Row_{dst}$
Cycle Detection

Propagation of d2 dependence:

\[ \text{Col}_{\text{dst}} = \text{Col}_{\text{src}} \]
\[ \text{Row}_{\text{src}} = \text{Row}_{\text{dst}} \]
\[ \text{Col}_{2} = \text{Col}_{1} \]
Cycle Detection

Propagation of $d_2$ dependence:

$Col_{dst} = Col_{src}$

$Row_{src} = Row_{dst}$

$Col_{2} = Col_{1}$
Cycle Detection

Propagation of d2 dependence:

Col_{dst} = Col_{src}
Row_{src} = Row_{dst}

Col2 = Col1
Cycle Detection

Propagation of d2 dependence:

Col_{dst} \neq Col_{src}
Row_{src} \neq Row_{dst}

Col_{2} \neq Col_{1}
Cycle Detection

Propagation of d2 dependence:
- Col\textsubscript{dst} = Col\textsubscript{src}
- Row\textsubscript{src} = Row\textsubscript{dst}
- Col2 = Col1
- Row1 = Row2

Xuehai Qian

BulkSMT: Designing SMT Processors for Atomic Block Execution
Cycle Detection

![Diagram showing cycle detection with T0, T1, and CT]
Cycle Detection

\[ T_0 \rightarrow d1 \rightarrow T_1 \]

CT
Cycle Detection

T0  T1

\[ \text{d1} \]

CT

\[ \text{d1} \]
Cycle Detection

\[ T0 \xrightarrow{d1} T1 \]

\[
\begin{array}{ccc}
  & & 1 \\
  & CT & \\
  d1 & & \\
\end{array}
\]
Cycle Detection

T0

T1

d1

d2

1

CT

d1
Cycle Detection
Cycle Detection

T0       T1

\[ \begin{array}{ccc}
  \text{d1} & \text{d2} & \text{d1} \\
\end{array} \]

\[ \begin{array}{ccc}
  \text{CT} & \text{1} & \text{CT} \\
\end{array} \]
Cycle Detection
Cycle Detection

T0

T1

d1

CT

d1

CT

d2

1

1

1

1

1
Cycle Detection

T0

T1

1

d2

1

1

1

d2

d1

CT

CT

CT

Xuehai Qian

BulkSMT: Designing SMT Processors for Atomic Block Execution

Thursday, December 8, 2011
Cycle Detection

Xuehai Qian

BulkSMT: Designing SMT Processors for Atomic Block Execution

Thursday, December 8, 2011
Cycle Detection

T0 | T1
---|---
dA | d1
   | d2
CT | CT
  | CT
1 | 1
  | 1
  | 1

Issues Related to CT and DT
Issues Related to CT and DT

- CT is not time-critical structure: is updated in the background
  - OK to find a cycle some time later (DT is always up-to-date)
- Update of CT is recursive
  - Terminates when any bit in diagonal is set or CT is no longer changed
- Always detect precise cycle, no false positive
# Hardware Mechanisms

<table>
<thead>
<tr>
<th>Mechanism</th>
<th>Function</th>
<th>Impl.</th>
<th>Designs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access Recording and Conflict Detection</td>
<td>Record the addresses accessed by each thread and detect when two threads have a data conflict</td>
<td>Access Bits in cache and related logic</td>
<td>All</td>
</tr>
<tr>
<td>Cycle Detection</td>
<td>Record data conflicts and their ordering, and detect conflict cycles</td>
<td>Dependence Table and Cycle Table</td>
<td>STALL ORDER</td>
</tr>
<tr>
<td>Advanced Conflict Recording</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Squash Set Generation</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Hardware Mechanisms

<table>
<thead>
<tr>
<th>Mechanism</th>
<th>Function</th>
<th>Impl.</th>
<th>Designs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access Recording and Conflict Detection</td>
<td>Record the addresses accessed by each thread and detect when two threads have a data conflict</td>
<td>Access Bits in cache and related logic</td>
<td>All</td>
</tr>
<tr>
<td>Cycle Detection</td>
<td>Record data conflicts and their ordering, and detect conflict cycles</td>
<td>Dependence Table and Cycle Table</td>
<td>STALL ORDER</td>
</tr>
<tr>
<td>Advanced Conflict Recording</td>
<td>Represent the type of conflict between different blocks compactly</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Squash Set Generation</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# Hardware Mechanisms

<table>
<thead>
<tr>
<th>Mechanism</th>
<th>Function</th>
<th>Impl.</th>
<th>Designs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access Recording and Conflict Detection</td>
<td>Record the addresses accessed by each thread and detect when two threads have a data conflict</td>
<td>Access Bits in cache and related logic</td>
<td>All</td>
</tr>
<tr>
<td>Cycle Detection</td>
<td>Record data conflicts and their ordering, and detect conflict cycles</td>
<td>Dependence Table and Cycle Table</td>
<td>STALL ORDER</td>
</tr>
<tr>
<td>Advanced Conflict Recording</td>
<td>Represent the type of conflict between different blocks compactly</td>
<td>Enhanced Dependence Table</td>
<td></td>
</tr>
<tr>
<td>Squash Set Generation</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Hardware Mechanisms

<table>
<thead>
<tr>
<th>Mechanism</th>
<th>Function</th>
<th>Impl.</th>
<th>Designs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access Recording and Conflict Detection</td>
<td>Record the addresses accessed by each thread and detect when two threads have a data conflict</td>
<td>Access Bits in cache and related logic</td>
<td>All</td>
</tr>
<tr>
<td>Cycle Detection</td>
<td>Record data conflicts and their ordering, and detect conflict cycles</td>
<td>Dependence Table and Cycle Table</td>
<td>STALL ORDER</td>
</tr>
<tr>
<td>Advanced Conflict Recording</td>
<td>Represent the type of conflict between different blocks compactly</td>
<td>Enhanced Dependence Table</td>
<td>ORDER</td>
</tr>
<tr>
<td>Squash Set Generation</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Advanced Conflict Recording

Enhanced Dependence Table

# Context

RAW
WAW
WAR

Enhanced Dependence Table

Xuehai Qian

BulkSMT: Designing SMT Processors for Atomic Block Execution

Thursday, December 8, 2011
## Hardware Mechanisms

<table>
<thead>
<tr>
<th>Mechanism</th>
<th>Function</th>
<th>Impl.</th>
<th>Designs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access Recording and Conflict Detection</td>
<td>Record the addresses accessed by each thread and detect when two threads have a data conflict</td>
<td>Access Bits in cache and related logic</td>
<td>All</td>
</tr>
<tr>
<td>Cycle Detection</td>
<td>Record data conflicts and their ordering, and detect conflict cycles</td>
<td>Dependence Table and Cycle Table</td>
<td>STALL ORDER</td>
</tr>
<tr>
<td>Advanced Conflict Recording</td>
<td>Represent the type of conflict between different blocks compactly</td>
<td>Enhanced Dependence Table</td>
<td>ORDER</td>
</tr>
<tr>
<td>Squash Set Generation</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# Hardware Mechanisms

<table>
<thead>
<tr>
<th>Mechanism</th>
<th>Function</th>
<th>Impl.</th>
<th>Designs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access Recording and Conflict Detection</td>
<td>Record the addresses accessed by each thread and detect when two threads have a data conflict</td>
<td>Access Bits in cache and related logic</td>
<td>All</td>
</tr>
<tr>
<td>Cycle Detection</td>
<td>Record data conflicts and their ordering, and detect conflict cycles</td>
<td>Dependence Table and Cycle Table</td>
<td>STALL ORDER</td>
</tr>
<tr>
<td>Advanced Conflict Recording</td>
<td>Represent the type of conflict between different blocks compactly</td>
<td>Enhanced Dependence Table</td>
<td>ORDER</td>
</tr>
<tr>
<td>Squash Set Generation</td>
<td>On a cycle of conflicts, decide the set of blocks to squash</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Hardware Mechanisms

<table>
<thead>
<tr>
<th>Mechanism</th>
<th>Function</th>
<th>Impl.</th>
<th>Designs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access Recording and Conflict Detection</td>
<td>Record the addresses accessed by each thread and detect when two threads have a data conflict</td>
<td>Access Bits in cache and related logic</td>
<td>All</td>
</tr>
<tr>
<td>Cycle Detection</td>
<td>Record data conflicts and their ordering, and detect conflict cycles</td>
<td>Dependence Table and Cycle Table</td>
<td>STALL ORDER</td>
</tr>
<tr>
<td>Advanced Conflict Recording</td>
<td>Represent the type of conflict between different blocks compactly</td>
<td>Enhanced Dependence Table</td>
<td>ORDER</td>
</tr>
<tr>
<td>Squash Set Generation</td>
<td>On a cycle of conflicts, decide the set of blocks to squash</td>
<td>Logic that operates on the Dependence Table</td>
<td></td>
</tr>
</tbody>
</table>
## Hardware Mechanisms

<table>
<thead>
<tr>
<th>Mechanism</th>
<th>Function</th>
<th>Impl.</th>
<th>Designs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access Recording and Conflict Detection</td>
<td>Record the addresses accessed by each thread and detect when two threads have a data conflict</td>
<td>Access Bits in cache and related logic</td>
<td>All</td>
</tr>
<tr>
<td>Cycle Detection</td>
<td>Record data conflicts and their ordering, and detect conflict cycles</td>
<td>Dependence Table and Cycle Table</td>
<td>STALL ORDER</td>
</tr>
<tr>
<td>Advanced Conflict Recording</td>
<td>Represent the type of conflict between different blocks compactly</td>
<td>Enhanced Dependence Table</td>
<td>ORDER</td>
</tr>
<tr>
<td>Squash Set Generation</td>
<td>On a cycle of conflicts, decide the set of blocks to squash</td>
<td>Logic that operates on the Dependence Table</td>
<td>ORDER</td>
</tr>
</tbody>
</table>
Set of Blocks to Squash on a Cycle

T0
wr

T1
rd
wr

T2
wr

RAW

WAW

WAR

Squash T0 ⇒ Squash T1
Squash T1 ⇒ Squash T0
Squash T0 ⇒ Squash T1
Squash T1 ⇒ Squash T0
Squash T0 ⇒ Squash T1
Squash T1 ⇒ Squash T0

Xuehai Qian

BulkSMT: Designing SMT Processors for Atomic Block Execution
Set of Blocks to Squash on a Cycle

T0  T1  T2
wr  rd  wr
wr  rd  wr

Squash T0 ⇒ Squash T1
Squash T1 ⇒ Squash T0

Xuehai Qian

BulkSMT: Designing SMT Processors for Atomic Block Execution

Thursday, December 8, 2011
Set of Blocks to Squash on a Cycle

- Squash the thread that closes the cycle (T0)
Set of Blocks to Squash on a Cycle

- Squash the thread that closes the cycle (T0)
- Forward propagate from T0
Set of Blocks to Squash on a Cycle

- Squash the thread that closes the cycle (T0)
- Forward propagate from T0
  - T0 ➔ T1: squash T1
Set of Blocks to Squash on a Cycle

- Squash the thread that closes the cycle (T0)
- Forward propagate from T0
  - T0 $\rightarrow$ T1: squash T1
  - T1 $\rightarrow$ T2: WAR, no propagate
Set of Blocks to Squash on a Cycle

- Squash the thread that closes the cycle (T0)
- Forward propagate from T0
  - T0 $\rightarrow$ T1: squash T1
  - T1 $\rightarrow$ T2: WAR, no propagate
- Backward propagate from T0
Set of Blocks to Squash on a Cycle

- Squash the thread that closes the cycle (T0)
- Forward propagate from T0
  - T0 \(\rightarrow\) T1: squash T1
  - T1 \(\rightarrow\) T2: WAR, no propagate
- Backward propagate from T0
  - T2 \(\rightarrow\) T0: RAW, no propagate
Outline

• Motivation
• Designing BulkSMT
  • Design Space
  • Hardware Mechanisms
• Multicore of BulkSMTs
• Evaluation
Multicore of BulkSMTs
Multicore of BulkSMTs

- Eager Scheme (E)
  - Block wants to commit
    - Commit globally first, then commit locally
  - Reception of cache invalidation
    - Check the cache access bits; may squash multiple blocks
- Lazy Scheme (L)
  - ......
Outline

• Motivation
• Designing BulkSMT
  • Design Space
  • Hardware Mechanisms
• Multicore of BulkSMTs
  • Evaluation
Evaluation

- Cycle-accurate execution-driven simulator based on SESC
- 6 SPLASH-2 and 2 PARSEC applications
- Applications run with 1, 4, or 16 threads
- Compare performance:
  - Using same number of cores, diff number of threads
  - Using same number of threads, diff number of cores
Execution Time (Same HW, 1 Core)

![Bar Chart]

- BK
- SQ
- ST
- OR

GeoMean

Execution Time

0

1.0

SQ, ST, OR

BK
• BulkSMT more cost effective: faster for the same HW
• BulkSMT ORDER is the best
Execution Time (Same HW, 4 Cores)

GeoMean

Execution Time

0

1.0

2.0

{SQ,ST,OR}→{LL,EE}

BK→{LL,EE}

GeoMean

Xuehai Qian

BulkSMT: Designing SMT Processors for Atomic Block Execution

Thursday, December 8, 2011
• BulkSMT ORDER is best
• BulkSMT limited by application scalability
Execution Time (Same Threads, 16)

GeoMean

Execution Time

{SQ,ST,OR}-{LL,EE}

BK-{LL,EE}

{SQ,ST,OR}-{LL,EE}

0

1.0

2.0

Xuehai Qian

BulkSMT: Designing SMT Processors for Atomic Block Execution
**Execution Time (Same Threads, 16)**

- BulkSMT multicore: using 1/4 hardware and achieves better performance
Also in the paper
Also in the paper

- Implementation issues
- Handling high-contention synchronization
- Other characteristics of execution behavior
- Related work
Conclusion
Conclusion

- Proposed **BulkSMT**: first SMT design that supports atomic-block (transactional) execution
  - Enables concurrency between dependent blocks
- Proposed designs of different concurrency vs cost
  - SQUASH on conflict
  - STALL on conflict
  - ORDER on conflict
- Designed a multicore of BulkSMTs
- BulkSMT is cost Effective
  - Higher performance for the same core count
  - Competitive performance for 1/4 of the cores
BulkSMT: Designing SMT Processors for Atomic-Block Execution

Xuehai Qian, Benjamin Sahelices and Josep Torrellas
University of Illinois

To appear at HPCA 2012

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