BulkSMT: Designing SMT Processors for Atomic-Block Execution

Xuehai Qian, Benjamin Sahelices and Josep Torrellas
University of Illinois

http://iacoma.cs.uiuc.edu/
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?
  - Enables a better utilization of the hardware
  - Fast communication between local contexts
  - Enables higher-concurrency forms of atomic block execution
Contributions

- **BulkSMT**: first SMT design with 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
  - Comparable performance for 1/4 of the cores
Outline

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

- Threads execute blocks of instructions atomically
- On inter-block dependence: typically squash and restart the block
Designing SMT For Blocked Execution

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

Contexts

L1/L2

spec rd

spec wr

No disp
SQUASH Design

Source Block

Destination Block

P0

LI$

L2$

wr x

rd x

wr x

squash

rd x

rd x
STALL Design

[Diagram showing the STALL design with instruction memory (L1$) and data memory (L2$), highlighting the stalls and commit process]
ORDER Design
Dependences in STALL

• On a dependence:
  • Stall the destination block
  • HW records source and destination blocks
• On commit/squash of source block
  • Wake up stalled destination block
Multiple Dependences in STALL

- Block can stall on an already stalled block: **Transitive Stall**

- Cycle not OK
  - When a stall cycle is formed, block that closes the cycle is squashed

```
T0 rw x
  |    |  \
|    |    |    \\
wr y

T1 rd x

T2
  |    |  \
|    |    |    \\
wr y

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

```
T1
  |    |  \
|    |    |    \\
wr y

T0 rw x
  |    |  \
|    |    |    \\
wr y

T1
  |    |  \
|    |    |    \\
wr y

Squash T1
Total order of blocks:
T0 → T1
```
Dependences in ORDER

- 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

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

Total order of blocks:
\[T1 \rightarrow T0 \rightarrow T2\]
Squash Policy in ORDER

- On a cycle: different dependences have different squash requirements

RAW

Squash T0 ⇒ Squash T1
Squash T1 ⇒ Squash T0

WAW

Squash T0 ⇒ Squash T1
Squash T1 ⇒ Squash T0

WAR

Squash T0 ⇒ Squash T1
Squash T1 ⇒ Squash T0

Friday, March 9, 2012
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>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>
Access Recording

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

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

Friday, March 9, 2012
## 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>

Friday, March 9, 2012
**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

---

**Diagram:**

- Nodes $Ti$ and $Tj$ connected by dependence $Ti \rightarrow Tj$
- Dependence Table (DT) with $n=4$
- Cycle Table (CT) with $n=4$
Dependence Cycle Detection

Propagation of d2 dependence:

Col2 ⊑ Col1
Row1 ⊑ Row2

Xuehai Qian

BulkSMT: Designing SMT Processors for Atomic Block Execution

Friday, March 9, 2012
Cycle Detection

![Diagram showing cycle detection process]

Xuehai Qian

BulkSMT: Designing SMT Processors for Atomic Block Execution

Friday, March 9, 2012
# 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>
Outline

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

- Global Eager Scheme (EE)
- Correct block commit:
  - Commit globally first, then commit locally
- Correct operation on reception of cache invalidation:
  - Check the cache access bits; may squash multiple blocks
- Global Lazy Scheme (LL): see paper
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 between BulkSMT and conventional block:
  - Using same number of cores, diff number of threads
  - Using same number of threads, diff number of cores
• BulkSMT more cost effective: faster for the same core count

• Among the BulkSMT designs, ORDER is the best
• BulkSMT ORDER is best
• BulkSMT limited by application scalability
Execution Time (Same Thread Count, 16)

- BulkSMT ORDER multicore:
  - Uses 1/4 of BK hardware and achieves comparable performance
  - Reason: ability to avoid squashes
Also in the paper

- Other hardware mechanisms
- Implementation issues
- Handling high-contention synchronization
- Other characteristics of execution behavior
- Related work
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 ORDER 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

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