BulkCompactor: Optimized Deterministic Execution via Conflict-Aware Commit of Atomic Blocks

Yuelu Duan, Xing Zhou, Wonsun Ahn, Josep Torrellas

University of Illinois at Urbana-Champaign
http://iacoma.cs.uiuc.edu

HPCA 2012, New Orleans, LA
Why Deterministic Execution?

• Shared-memory multiprocessing is nondeterministic
  – Hard to reproduce execution/error
  – Complicates concurrent programming, testing and debugging

• Deterministic Execution
  – Given an input, always produces same result
  – Enforces an identical thread interleaving at each run
Current Deterministic Execution Models

- Parallel execution of *chunks* w/o communication
- Lazy data merge in-order (or *round-robin*)

- P1
- P2
- P3

- Access private buffer
- Commit

- Commit token
Current Deterministic Execution Models

- lazy data merge in-order (or *round-robin*)

On identifying a **Data Conflict** during the merge:
- Ignore it
  + Fast
    - Unintuitive memory model -- hard to reason
Current Deterministic Execution Models

On identifying a **Data Conflict** during the merge:

- Honor the dependence --> squash the chunk
  - Intuitive memory model
  - Becomes **performance bottleneck**

lazy data merge in-order (or *round-robin*)

Understand & Fix this!
Problem in Round Robin: Transitive Squash Delay

**Transitive Squash Delay**: On conflict & squash, all the successor processors observe the squashed work overhead.

Multiple transitive squash delays accumulate

- With more processors:
  - Squash rate of each processor increases
  - More processors cause transitive squash delay
  - Overhead increases as $O(N^2)$
Our Contributions

• **BulkCompactor: Conflict-Aware Deterministic Commit**
  – Avoid transitive squash delay

![Diagram showing P1, P2, P3, P4, with arrows indicating removing transitive delay and compacted execution.](image)
Our Contributions

• BulkCompactor: Conflict-Aware Deterministic Commit
  – Avoid transitive squash delay
• Design of a Centralized & Distributed Commit Machine
• Results for 32 processor runs
  – Only 22% execution overhead over a nondeterministic system
  – Overhead scalable as $O(N)$
  – Conventional round-robin deterministic execution: 133% overhead and $O(N^2)$
Conflict-Aware Commit

Key Idea:
Commit token *bypasses* a squashed chunk, and *postpones* it to a new round
Conflict-Aware Commit

Benefits:
- A squashed chunk does not cause transitive squash delay
- A conflict-free chunk can proceed to commit regardless of other's behavior
Squashes are not Deterministic

P2 is squashed & postponed

P2 is not squashed, but **must** be postponed!

➤ Dependences are Deterministic!
Enforcing the Dependences

Non-Postponed Set:
Set of write addresses in the committed chunks
Enforcing the Dependences

1. P1 \rightarrow P2
2. P3 \rightarrow P4
3. \emptyset \rightarrow P2
4. P2 \rightarrow P3
5. P4 \rightarrow P4
Enforcing the Dependences

• Limitation:
  – Non-Postponed Set may become very large
  – \( O(N) \) chunks are compared for each commit
**BulkCompactor-S: Scalable Postponement**

- Dependences between $M$ consecutive chunks:
  - As usual: cause chunk postponement
- Dependences beyond $M$ consecutive chunks:
  - Cause squash and re-execution in critical path

\[ M = 1 \]
BulkCompactor-S Operation

$M = 1$

1. $\emptyset$ from P1 to P2
2. $\emptyset$ from P2 to P3
3. $\emptyset$ from P3 to P4
BulkCompactor-S Operation

$M = 1$

- $P_1$
- $P_2$
- $P_3$
- $P_4$

Steps:
1. Transition from $P_1$ to $P_2$
2. Transition from $P_3$ to $P_2$
3. $P_4$
4. $\emptyset$
5. Transition from $P_2$ to $P_3$
BulkCompactor-S

• Advantage:
  – Scalable: only $O(M)$ chunks are compared
  – If most conflicts are between neighbor processors (conflict-locality) --> avoid most transitive delays

• Disadvantage:
  – Some transitive squash delay are not avoided
Outline

• BulkCompactor & BulkCompactor-S
• Design of a Centralized & Distributed Commit Machine
Centralized Commit Machine

- Extensions to BulkSC[ISCA'07]
- Postponement Detector Module
  - Keeps Non-Postponed Set and commit token
  - Placed together with commit arbiter
  - Uses signatures to encode addresses in Non-Postponed Set and to perform conflict detection
Complete Commit/Postponed Process

P1 \rightarrow \text{(R1, W1)} \rightarrow \text{Postponement Detector} \rightarrow \text{Arbiter} \rightarrow \text{Directory}

P2

P3
Complete Commit/Postponed Process
Complete Commit/Postponed Process

P1

Squashed by P1

P2

Postponement Detector

W1

W1

Arbiter

P3

Directory
Complete Commit/Postponed Process

Duan, Zhou, Ahn, Torrellas

BulkCompactor: Conflict-Aware Deterministic Execution
Complete Commit/Postponed Process

- P1
- P2
- P3

Postponement Detector
- W1

Arbiter
- W1

Directory
Complete Commit/Postponed Process

- P1
- P2
- P3

Postponement Detector
- W1

Arbiter

Directory
Complete Commit/Postponed Process

W1 ∩ (R3, W3) = Ø ?
Complete Commit/Postponed Process

\[ W1 \cap (R3, W3) = \emptyset \text{? NO!} \]

- P1
- P2
- P3
- Postponement Detector
  - W1
- Arbiter
- Directory
Distributed Commit Design

Distributed P-detectors, each handle a range of addresses

P1  P2  P3  P4

D1  D2  D3  D4

See the paper for details..
Issues in Deterministic Chunk Execution

• Fix the chunk size
• Use modest sized chunk and enough buffer
  – Avoid cache overflow (nondeterministic)
• Process interrupts at chunk boundaries
• On an exception, re-execute the chunk only when the token arrives
Outline

- BulkCompactor & BulkCompactor-S
- Design of a Centralized & Distributed Commit Machine
- Evaluation
Evaluation

- Model the Bulk Multicore with 4 to 32 processors
  - Architecture uses chunks with lazy commit [CACM'09]
  - 2k instruction chunk size
  - Model Round-robin, BulkCompactor, BulkCompactor-S4, BulkCompactor-S8 commit

- Simulation based on PIN

- Apps from PARSEC and SPLASH2
  - Run to completion
• Round-robin incurs high overhead (~2.5x)
• BulkCompactor only slightly slower than nondeterministic system
• BulkCompactor-S4 & S8 perform like BulkCompactor for many apps
  – conflict locality!

**Duan, Zhou, Ahn, Torrellas**

*BulkCompactor: Conflict-Aware Deterministic Execution*
Performance Scalability (f_{mm})

- Overhead of Round-robin grows fast
- BulkCompactor grows almost linearly
Also in the paper..

• Overhead analysis for each scheme
• Handling starvation and fairness
• Conflict behavior characterization
• Sensitivity to the chunk size
• Sensitivity to the token passing latency
Conclusions

• Presented **BulkCompactor: Conflict-Aware Deterministic Commit System**
  – Avoids transitive squash delay
  – Performs comparably to nondeterministic chunk-based system
  – Scalable

• Designed centralized and distributed commit machine

• Results for 32 processor runs
  – Only 22% execution overhead over a nondeterministic system
BulkCompactor: Optimized Deterministic Execution via Conflict-Aware Commit of Atomic Blocks

Yuelu Duan, Xing Zhou, Wonsun Ahn, Josep Torrellas

University of Illinois at Urbana-Champaign
http://iacoma.cs.uiuc.edu

HPCA 2012, New Orleans, LA
Backup Slides
Squash is not Deterministic (2)

C₃ is neither squashed nor postponed

C₃ is squashed, but **must not** be postponed
Rules of Deterministic Postponement

1. *Always* postpone a chunk if it conflicts with previously committed chunks in the same round

2. *Never* postpone a chunk if it conflicts with committed chunks from earlier round
BulkCompactor vs. BulkCompactor-S

BulkCompactor

BulkCompactor-S \( (M = 1) \)
Signatures Sent in Parallel

Distributed P-detectos, each handle a range of addresses

Signatures sent to relevant P-detectos, with a leader [MICRO'10]

Duan, Zhou, Ahn, Torrellas

BulkCompactor: Conflict-Aware Deterministic Execution
Passing Commit Token

Distributed P-detectors, each handle a range of addresses

Token (Bit vector): indicates each Pi being postponed or not
Deterministic Conflict Detection

• Signature operations can introduce false positive conflicts
  – Use virtual address to encode signatures
    • Identical across program runs
  – Use same encoding algorithm across machines

• In distributed BulkCompactor:
  – Index p-detectors/directory deterministically