Exploiting Commutativity to Reduce the Cost of Updates to Shared Data in Cache-Coherent Systems

Guowei Zhang, Webb Horn, Daniel Sanchez

MICRO 2015
Executive summary

- Updates to shared data limit parallelism in current systems

- Insight: Many updates are commutative

- Coup extends cache coherence protocols to make commutative updates as cheap as reads
  - Maintains coherence and consistency
  - Accelerates update-heavy applications significantly
Updates are expensive

Shared cache

Core 0
add(A, 1);
add(A, 1);
read(A);

Core 1
add(A, 2);
add(A, 2);
add(A, 2);

Traffic
Serialization
Updates are expensive, even with RMOs

Traffic
Serialization
Complicates consistency
Coup: exploiting commutativity

Low traffic
Concurrent updates
Simple consistency

Less general than RMOs
Commutative updates are common

- Operations
  - +
  - ×
  - MIN
  - OR

- Applications
  - Reduction variables
  - Iterative algorithms
  - Graph traversal
  - Reference counting
Software privatization vs. Coup

Software privatization

Needs to amortize privatization/reduction costs

Wastes shared cache & memory capacity

Must apply selectively

Coup

No overheads

No wasted capacity

Apply to any update that might commute

One read-only copy

Multiple thread-private, update-only copies
Outline

- Introduction
- Coup
- Evaluation
Structural changes

- Shared cache/dir
- Reduction unit
- Coherence states (M, S, I, ... U)
- ISA:
  - comm_add (&x, v)
  - comm_or (&x, v)
  - ... load (&x)
  - Store (&x, v)
  - ...

Private Cache 0 → Private Cache N-1
Core 0 → Core N-1
Example: extending MSI

Transitions
- Initiated by own core (gain permissions)
- Initiated by others (lose permissions)

Legend
- States: Modified, Shared (read-only), Invalid, Update-only
- Requests: Read, Write, Commutative update

Graph showing state transitions and operations for MSI and MUSI.
Coherence and consistency

- Coherence is maintained

- Consistency is not affected

- See paper for proofs
Implementation and verification

No extra stable states

Easy to verify
Evaluation Methodology

1-8 processor and L4 chips

Processor chip organization

Shared L3 and chip directory

L2 0
L1I L1D
Core 0

L2 15
L1I L1D
Core 15

to L4 chips

8 sockets × 16 cores/socket = 128 cores
Coup vs. Atomic Operations

Fraction of commutative instructions

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>MESI</th>
<th>COUP</th>
</tr>
</thead>
<tbody>
<tr>
<td>histogram</td>
<td>1.0%</td>
<td>2.4%</td>
</tr>
<tr>
<td>spmv</td>
<td>4.9%</td>
<td>0.40%</td>
</tr>
<tr>
<td>pagerank</td>
<td>0.96%</td>
<td></td>
</tr>
</tbody>
</table>

Normalized AMAT
## Modifying algorithms to exploit Coup

### Delayed deallocation reference counting

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Data structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Refcache$^{[1]}$</td>
<td>Hash table</td>
</tr>
<tr>
<td>Coup implementation</td>
<td>Hierarchical bit vectors + comm add/or</td>
</tr>
</tbody>
</table>

![Bar chart comparing performance between Refcache and Coup](chart.png)

Conclusions

- Coup allows concurrent commutative updates
  - Maintains coherence and consistency

- Coup implementation accelerates single-word updates
  - Minor hardware overhead
  - Accelerates update-heavy applications by up to 2.4x

- Coup opens exciting research avenues
  - Commutativity-aware hardware transactional memory
  - Support arbitrary update functions, semantic commutativity
THANKS FOR YOUR ATTENTION!

QUESTIONS ARE WELCOME!