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

Guowei Zhang, Webb Horn, Daniel Sanchez
(zhanggw, webbhorn, sanchez)@csail.mit.edu

1. Motivation

Updating shared data limits performance
- Significant traffic and serialization

Opportunity: many updates are commutative
- Same final result regardless of order
- Operations: add, mul, min, max, bitwise logical...
- Applications: reduction variables, iterative algorithms, graph traversal, reference counting...

Commutativity has been widely exploited in software, but hardware still understands only reads and writes!

2. Updates are expensive

Remote memory operations
- Global traffic and serialization
- Complicated consistency model
- Reduced data movement

3. Coup: Exploiting commutativity

- Local and concurrent commutative updates
- Reduction upon reads

4. Coup vs. prior work

Delegation: e.g., send all updates to a single thread
Privatization: e.g., update thread-local copies

Compared to RMOs:
- less general, but much better as long as reusing data

Compared to privatization:
- fast read/update switches, no wasted capacity

5. Example: Extending MSI

Coup can be applied to any conventional invalidation-based coherence protocol!

6. Implementation and verification

- Reads are another type of commutative update
- Generalize shared state to non-exclusive state

Coup requires NO extra stable states and 1-5 extra transient states

- Can be verified up to a large number of commutative updates with reasonable cost

7. Coherence and consistency

Coup preserves cache coherence
- Single-writer, multiple-reader is not maintained

Coup does not affect consistency
- E.g., applying store fences to commutative updates

8. Evaluation against atomic ops

9. Evaluation against privatization

Case study: reduction variables

Case study: reference counting