Is SC + ILP = RC?

Presented to CS258 on 3/12/08
by David McGrogan

Review: Acronym Salad

- SC: Sequential Consistency
  - Memory operations occur in program order
- ILP: Instruction Level Parallelism
  - Operations moved around in hardware
- RC: Release Consistency
  - Memory operations may be re-sequenced except across releases of synch. variables

Ease vs. Performance

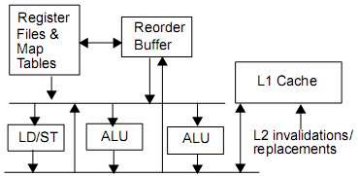
- Sequential consistency
  - Intuitive for programmer
  - Limits performance-enhancing hardware
- Release consistency
  - Requires manual classification of mem. access
  - Provides excellent performance

Improving SC, part 1

- SC requires only that the result be equivalent to operations in program order
- Adding ILP to SC allows better performance

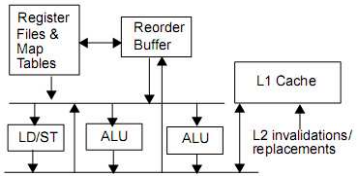
How it works

- Load/store queue maintains program order
- Exception/mispredict causes reorder buffer rollback
- Queued block prefetch
- Loads reordered



What about stores?

- In standard SC, reorder buffer blocks on loads if a store is pending
- Pipeline can stall on long stores as buffer/ queue fill



The RC ideal

- Most memory accesses can overlap in any given order
- Store buffering to bypass pending stores
- Binding prefetches – fetch before load reaches head of reorder buffer
- Speculative relaxation even across fences

Improving SC, part 2

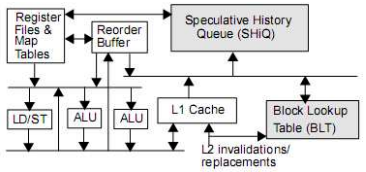
- SC requires only that the result be equivalent to operations in program order
- Anything allowed as long as this is upheld
- Let's go nuts

Rampant Speculation

- SC++ relaxes all memory access orders
- State changes after pending stores kept in history, can be undone if necessary
- Undo happens on external coherence action
- Impossible without additional hardware

Additional Hardware

- Speculative History Queue (SHiQ) logs modifications to processor state and L1 cache, permits undo to oldest pending store
- Spec. stores done on L1 cache, not queued
- BLT speeds detection of rollback necessity



Avoiding Deadlock

- After rollback, all pending stores must complete before speculation can resume
- Coherence handling paused during rollback

Sources of High Rollback

- Data races in application
- Significant false sharing
- Inevitable cache conflicts

- A DSM system won't help apps like this no matter what; communication dominates

Performance

- Better than SC, about as good as RC

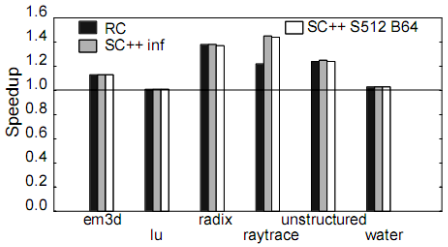


FIGURE 3: Comparison of SC, RC, and SC++.

Now with 4x net latency

- Large SHiQ allows SC to handle laggy networks

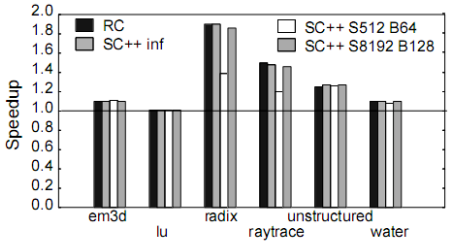


FIGURE 4: Impact of network latency.

Is SC++ really necessary?

- A huge reorder buffer doesn't save SC in every case

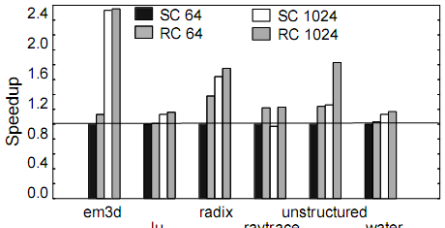


FIGURE 5: Impact of reorder buffer size.

Speculative Stores

- Not always important, but sometimes very significant

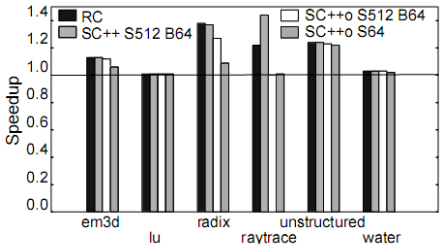


FIGURE 6: Impact of speculative stores.

The Importance of L2

- Too small, and the miss rate goes up
- Some apps cause many conflict misses

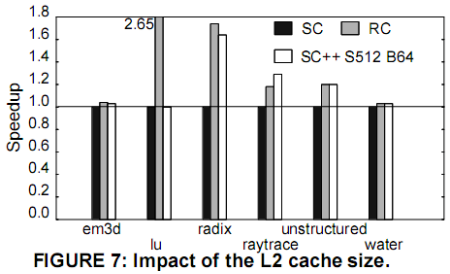


FIGURE 7: Impact of the L2 cache size.

Wrapup

- SC++ achieves its gains by mimicking RC
- Maintains convenience to programmer at expense of added hardware
- Later results provide further improvement