Speculative Lock Elision: Enabling Highly Concurrent Multithreaded Execution

Ravi Rajwar and Jim Goodman
University of Wisconsin-Madison

International Symposium on Microarchitecture, Dec. 2001

Funding Ack.: NSF grant CCR-9810114 and Intel Corporation

Elision:	the act or an instance of dropping out or omitting something
Elide :	to leave out of consideration

The problem

- **Locks**
  - Contention limits performance
  - Contention limits scalability
  - Cause unnecessary memory traffic
  - Involve long latency memory accesses
- **Programmers can use finer-grained locks**
  - Time-consuming
  - Difficult and Error-prone

The question

Can speculative hardware provide?

- Higher performance
- More scalability
- Without software changes

3

The solution

Speculative Lock Elision

Idea

1. "Get" lock without writing lock variable
2. Execute critical section speculatively
3. Track data (blocks) read/written
4. Rollback when data conflicts occur

Features

- Implement with small changes to processor
- Transparent to software
- Use existing coherence protocol for tracking data
- **Non-conflicting critical sections execute in parallel!**

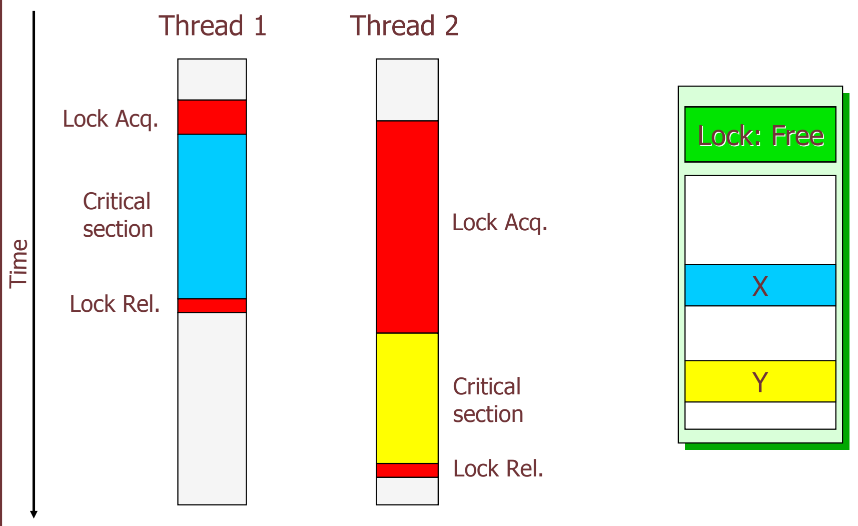
4

Outline

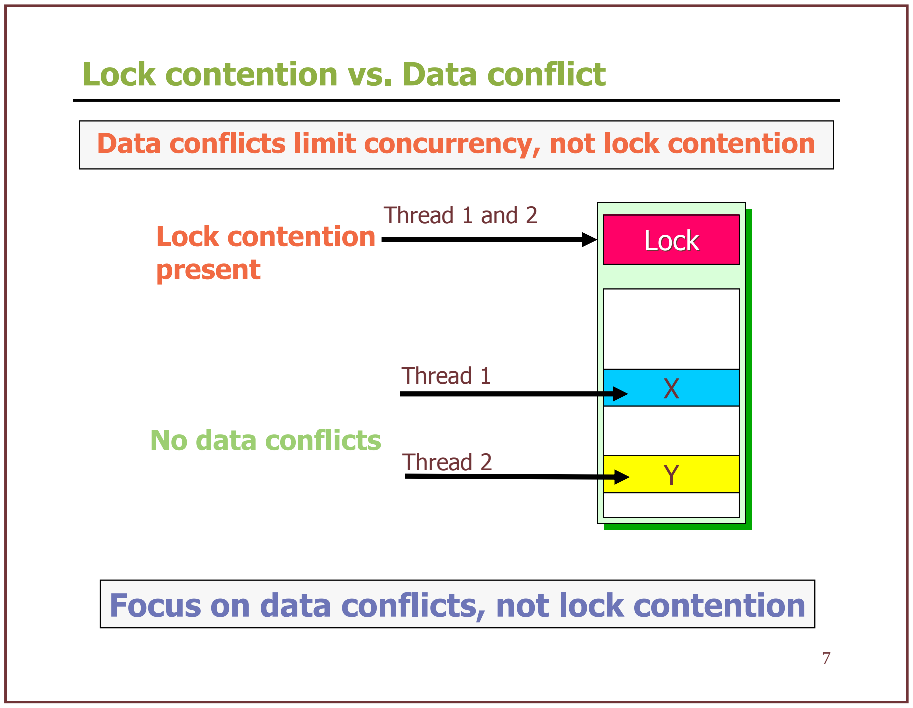
- Motivation
  - Thread-safe hash table
  - Lock contention vs. data conflict
- Speculative Lock Elision concept
- Speculative Lock Elision details
- Results
- Concluding remarks

5

Conservative lock use: thread safe hash table

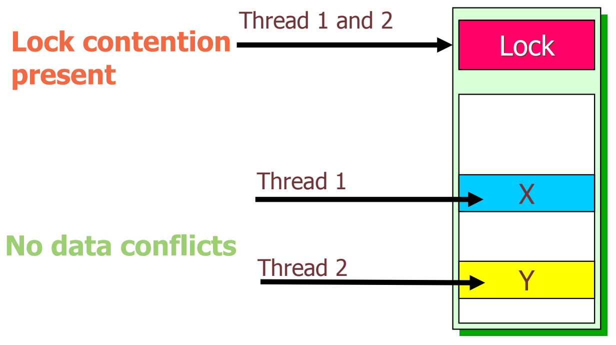


6



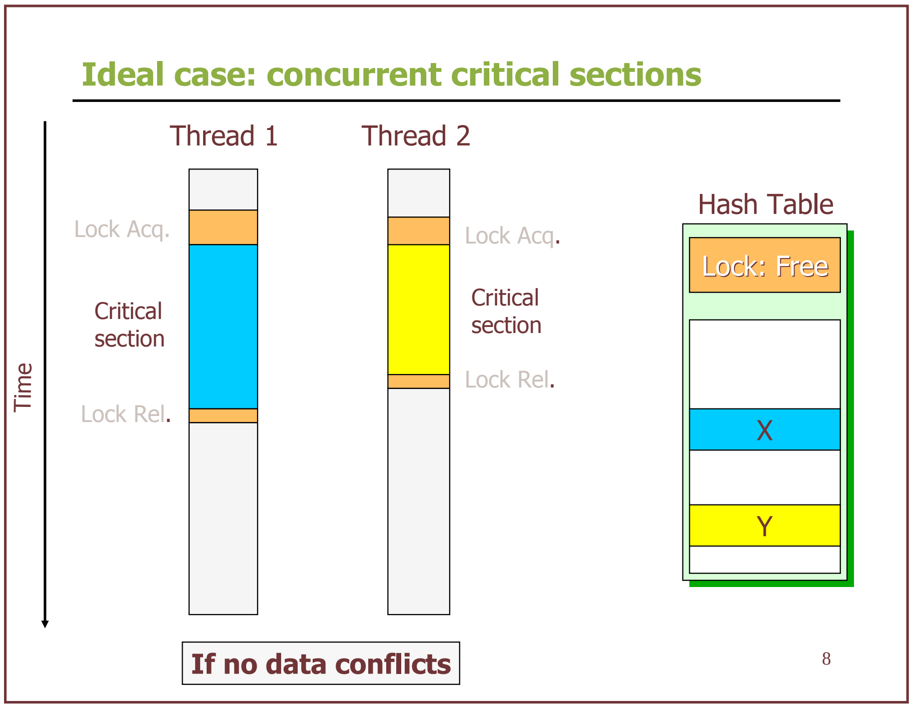
Lock contention vs. Data conflict

Data conflicts limit concurrency, not lock contention

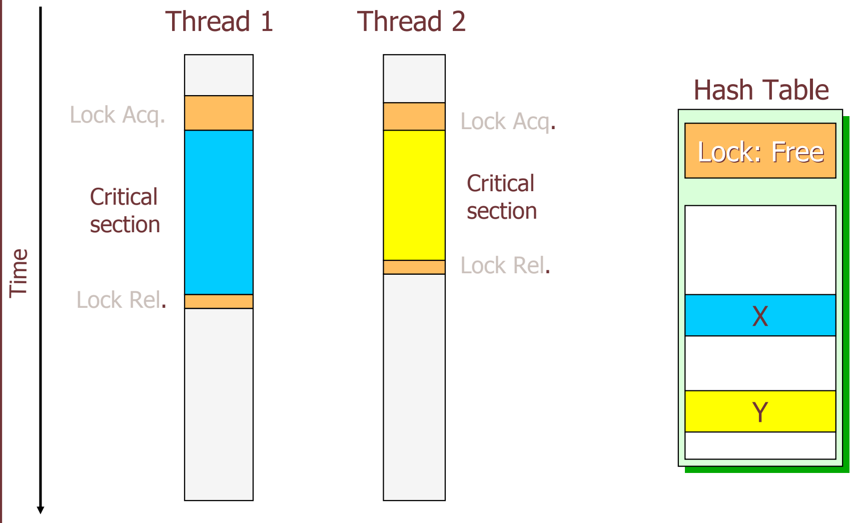


Focus on data conflicts, not lock contention

7



Ideal case: concurrent critical sections



If no data conflicts

8

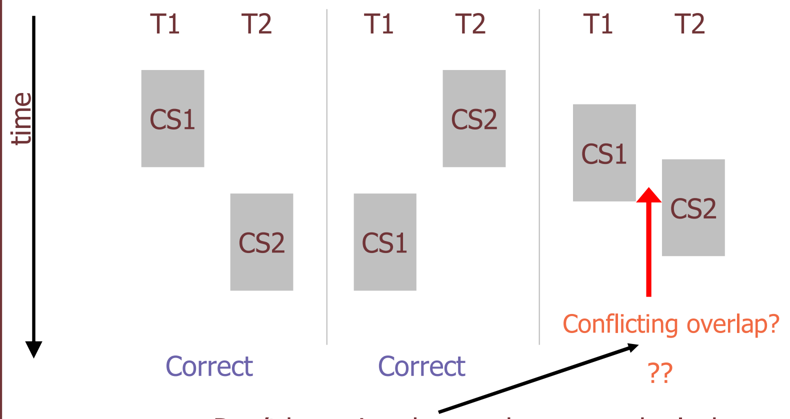
Outline

- Motivation
- Speculative Lock Elision concept
  - Critical sections
  - Conceptual view of SLE
- Speculative Lock Elision details
- Results
- Concluding remarks

9

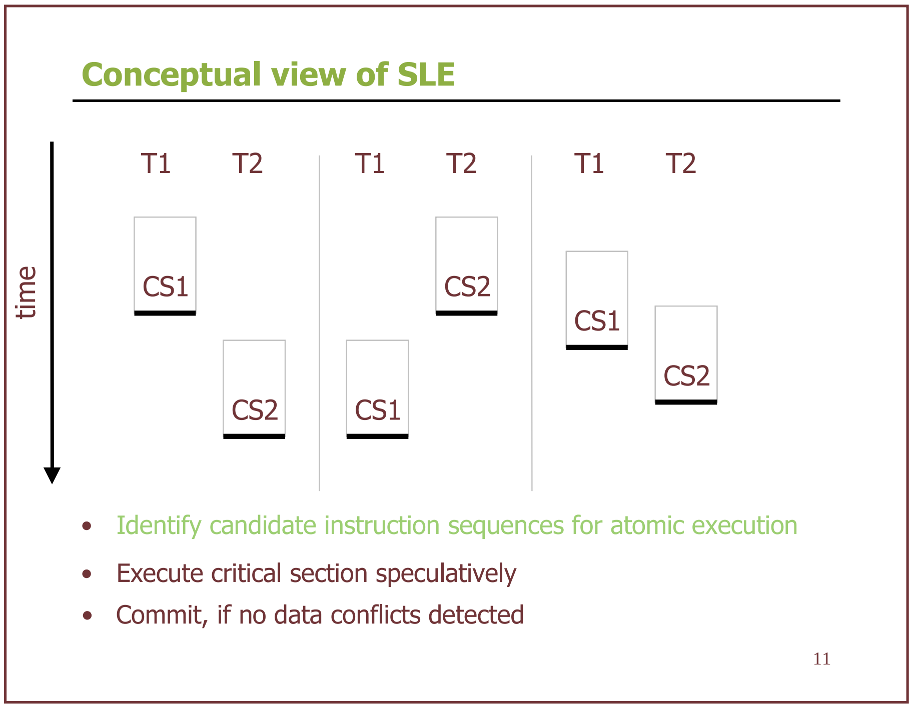
Critical sections

- **Provide atomicity**
- Implemented using locks

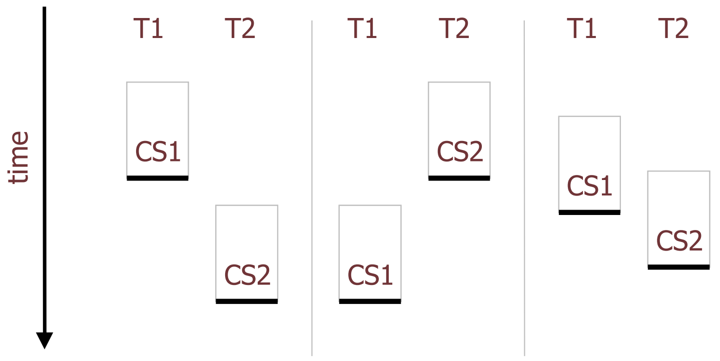


Don't know in advance, hence need a lock

10

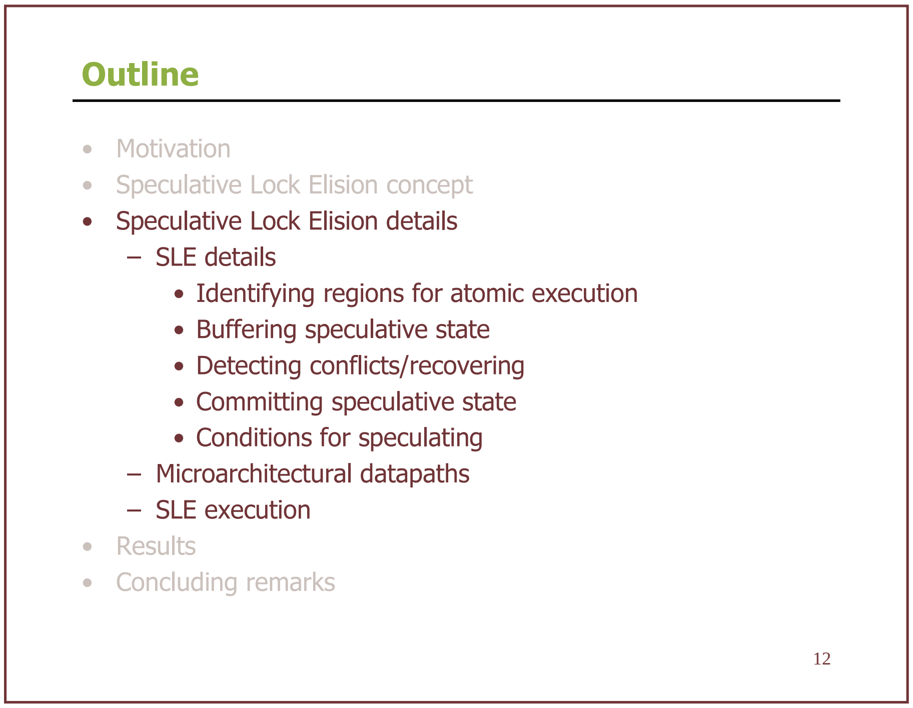


Conceptual view of SLE



- Identify candidate instruction sequences for atomic execution
- Execute critical section speculatively
- Commit, if no data conflicts detected

11



Outline

- Motivation
- Speculative Lock Elision concept
- Speculative Lock Elision details
  - SLE details
    - Identifying regions for atomic execution
    - Buffering speculative state
    - Detecting conflicts/recovering
    - Committing speculative state
    - Conditions for speculating
  - Microarchitectural datapaths
  - SLE execution
- Results
- Concluding remarks

12

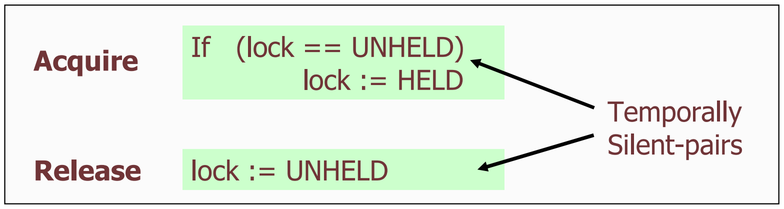
Identifying regions for atomic execution

- **Look for instruction sequences to execute atomically**
- **Locks provide a good hint**
  - Protect critical sections
- **Hardware normally cannot identify lock operations**
  - Can predict these operations
- Predict "candidate instruction sequence" for execution
  - Exploit property of lock variables

13

Key: Lock variables exhibit temporal silence

- **Exploit property of lock variables**
  - Lock variables exhibit temporal silence
  - Lock release **restores value of lock** prior to lock acquire



Acquire

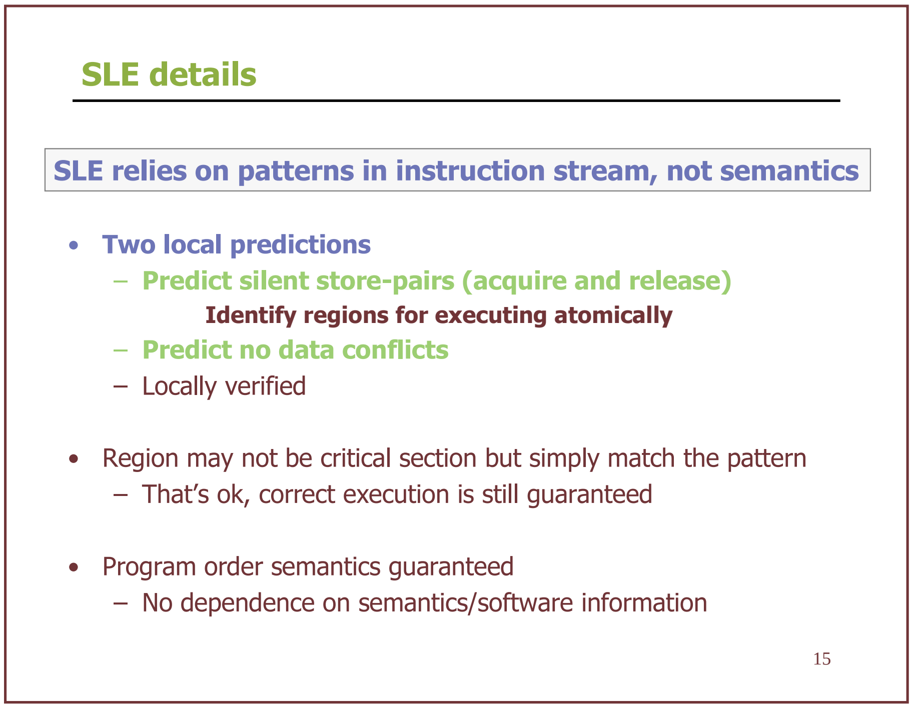
```
If   (lock == UNHELD)
          lock := HELD
```

Release

```
lock := UNHELD
```

- Speculating thread sees acquired lock
- Other threads see a free lock

14



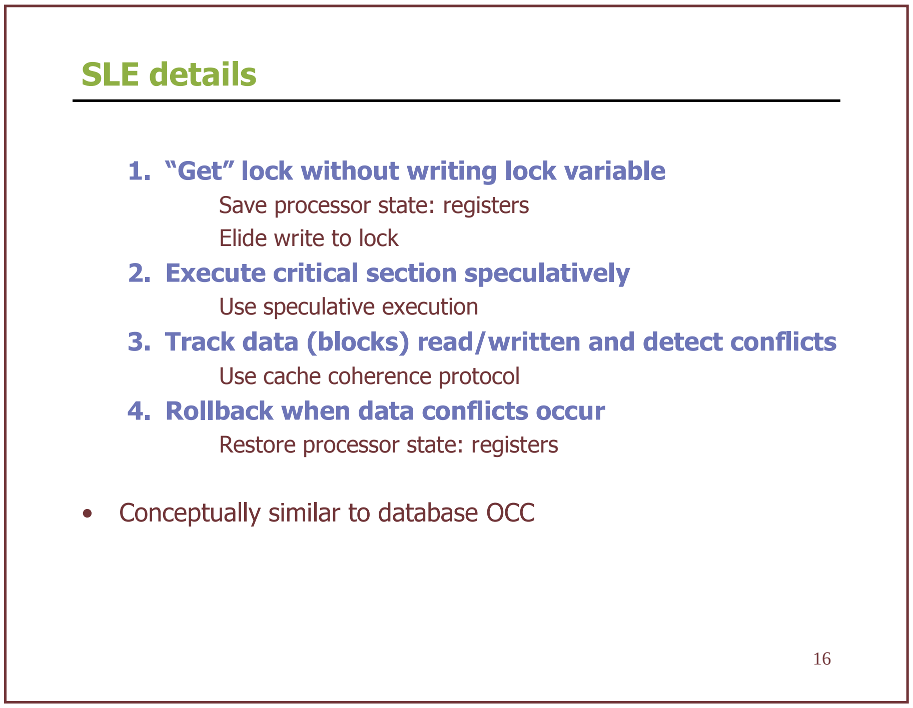
SLE details

SLE relies on patterns in instruction stream, not semantics

- **Two local predictions**
  - **Predict silent store-pairs (acquire and release)**
    - **Identify regions for executing atomically**
  - **Predict no data conflicts**
  - Locally verified

- Region may not be critical section but simply match the pattern
  - That's ok, correct execution is still guaranteed

- Program order semantics guaranteed
  - No dependence on semantics/software information



SLE details

1. **"Get" lock without writing lock variable**
   Save processor state: registers
   Elide write to lock
2. **Execute critical section speculatively**
   Use speculative execution
3. **Track data (blocks) read/written and detect conflicts**
   Use cache coherence protocol
4. **Rollback when data conflicts occur**
   Restore processor state: registers

- Conceptually similar to database OCC

Buffering speculative state

- **Register state**
  - No need to track inter-instruction dependences
  - One checkpoint already available for restoring
  - Speculatively retire instructions

- **Memory state**
  - Augment write buffer to store speculative data
    - Any other buffering technique ok
  - Speculative data never exposed until commit

17

Detecting data conflicts

- Data conflicts
  - For any two threads, at least one is writing the data block
  - Extend cache with speculative access bits
    - Track data block accesses
- **Coherence protocol provides functionality**
  - Writes to shared locations generate invalidations
  - Zero cost detection
- Lock kept in shared state
  - Automatically notified if anyone acquires lock
  - Misspeculation may result in explicit lock acquisition

18

Recovering from misspeculation

- **Recovery**
  - **Treat like a branch misprediction**
  - Roll-back processor state
  - Squash speculative entries in write buffer
  - Coherence protocol does not care
- Restart threshold N
  - Can re-attempt speculation until N failures

19

Committing speculation

- **Register state**
  - Simply discard checkpoint
- **Memory state**
  - Write permissions for modified blocks already obtained
    - Write requests (not data) exposed to outside world
    - Write buffer is marked as having latest data
    - Alternatively, can use cache to buffer data
  - Coherence state transitions support present
    - Speculative loads, exclusive prefetches
  - **Provides illusion of atomic commit of all writes**

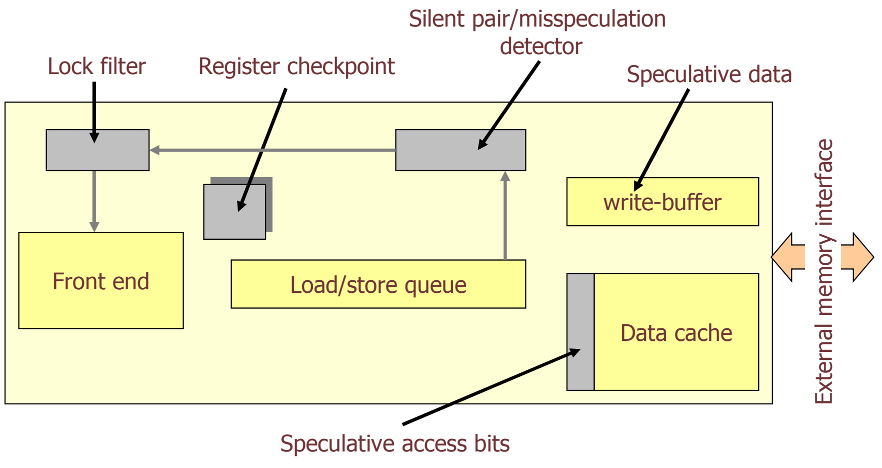
20

SLE summary

- **Conditions for starting speculation**
  - Appropriate starting instruction/value pattern detected
- **Conditions for ending speculation**
  - Restoring store to lock detected
  - Non-restoring store to lock detected
  - Some operation cannot be performed speculatively
  - Appropriate terminating pattern not detected
  - Data conflict detected
- **All operations between start and end appear atomic**
- **Memory consistency**
  - Always correct to insert an atomic set of memory operations into a global order of operations under any memory model

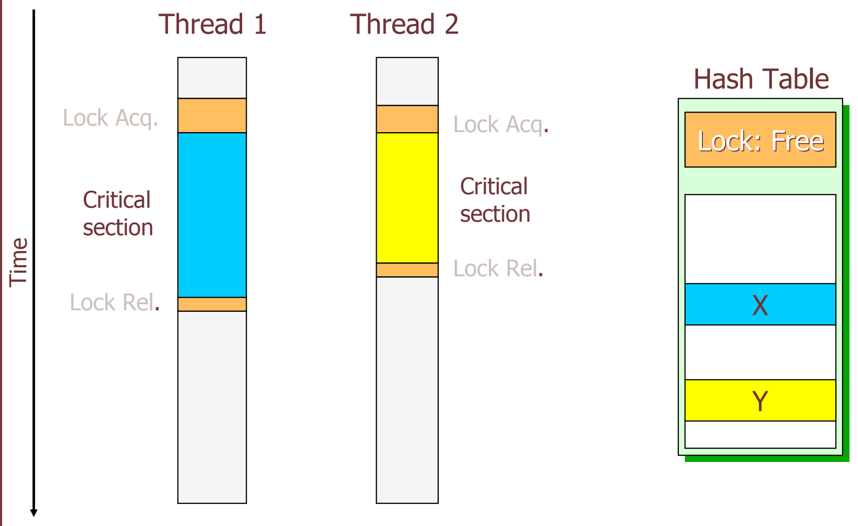
21

Microarchitecture data-paths



22

SLE execution

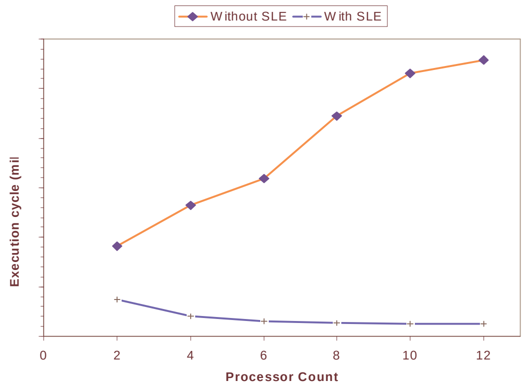


Outline

- Motivation
- Speculative Lock Elision concept
- Speculative Lock Elision details
- Results
- Concluding remarks

Micro-benchmark result

- CMP, N counters, N processors (one per counter), 1 lock
- 2^{16}/N increments per processor



25

Application results

- CMP/SMP/DSM configurations for 8 and 16 processors
- Sun Gigaplane-type bus and SGI Origin 2000-type directory
- Splash/Splash2 applications used
- Detailed parameters in paper

26

Outline

- Motivation
- Speculative Lock Elision concept
- Speculative Lock Elision details
- Results
- Concluding remarks
  - SLE advantages/limitations
  - SLE summary

27

SLE advantages

Implementation

+ Uses well understood speculation techniques
+ Much functionality already present: coherence protocol
+ **Transparent to programmers**
  No software or instruction set support
+ No system level changes
  Completely in the micro-architecture
  No coherence protocol changes

Performance

+ **Lock never written to**
  No serialization on lock
  Kept locally in shared state
+ Concurrent execution
+ Reduced observed memory latencies
+ Reduced memory traffic

28

SLE limitations

- – Extra data paths and hardware
- – Lock acquired for misspeculation conditions other than conflicts
  - Resource constraints, I/O, certain types of system calls
  - Get same performance and behavior as without SLE
- – Sensitive to restart threshold
  - Coherence protocol interference

29

SLE contributions

- **Enables highly concurrent multithreaded execution**
  - Concurrently execute non-conflict critical sections
  - Validation without writing to/acquiring lock
- **Simplifies correct multithreaded code development**
  - Programmers do not have to learn new techniques
- **Easy implementation**
  - Does not require software or instruction set support
  - Minor changes to modern processors
- **Opens up optimization opportunities…**
  - – Can provide strong performance guarantees even with conflicts?

Simulation parameters for SMP

Processor speed	1 GHz (1 ns clock)
Reorder buffer	128 entry with a 64 entry LSQ
Issue mechanism	Out-of-order issue/commit of 8 ops. per cycle
Branch predictor	8-K entry combining predictor, 8-K 4-way BTB
Write buffer	64-entry
L1 instruction cache	64-KB, 2-way, 1-cycle access, 8 pend. misses
L1 data cache	128-KB, 4-way, write-back, 3 ports, 1-cycle access, 8 pen. misses
L2 unified cache	4-MB, 4-way, write-back, 12 cycle access, 16 pend. misses
Line size	64 bytes
Coherence protocol	Sun Gigaplane-type MOESI protocol b/w L2s. Split transaction.
Address network	Broadcast snooping, 20 ns per snoop, 120 outstanding transactions.
Data network	Point-to-Point, pipelined, 70 ns transfer latency
DRAM memory module	8-bytes wide, ~70ns access time for 64 byte line
Memory consistency	Total Store Ordering

More configurations in paper

backup

Related work

- Concurrency techniques
  - Lamport
  - Transactional memory
  - Load-Linked/Store-conditional
- Database techniques
  - Optimistic concurrency control
- Speculative buffering
  - ARB (Franklin & Sohi)
  - SVC (Gopal et al.)
  - Speculative Retirement (Ranganathan et al., Lai & Falsafi)
  - Multiversion Memory (Sorin et al.)

backup

Predicting critical sections

- Need only detect atomic TEST&SET operation
  - Load-locked/store-conditional sequence
  - No need to detect TEST operations
- Track stores to address of LL/SC operand to detect lock release
- PC of store-conditional used to identify critical section

- Similar approach for other primitives

backup

Nested critical sections

- Does not matter
  - Automatically handled
  - Speculation un-aware of critical sections
- Commit a sequence of instructions atomically
- The sequence can have nested critical sections too
  - Speculation does not care about locks or nesting
  - Speculation simply looks for sequence to commit atomically
- For a dynamic instance of speculation
  - All "critical sections" nested within are executed normally

backup