Optimizing Synchronization

18742-Computer Architecture and Systems

Ashish Dwivedi, Deepali Garg

Electrical and Computer Engineering
Carnegie Mellon University

February 6, 2020

Electrical & Computer ENGINEERING

Schedule

Electrical & Computer ENGINEERING

Overview

Overview

Similarities

1. Both the papers recognize locking and critical sections inside locks as a major bottleneck in speeding up the parallel processor's speed.
2. Both the papers do not propose any ISA update
3. Both the papers quote (though not quantitatively) that HW update is minimal

Differences

1. ***Speculative Lock Elision*** paper recognizes that most of the time locking is not required for correct execution of the program hence elision could speed up the system
2. ***IQL + SP*** paper recognizes that lock requests can be queued and hence speculative forwarding of lock and critical data has the potential to speed up the system

Electrical & Computer ENGINEERING

Speculative Lock Elision

Motivation and Contribution

- Serialization of threads due to critical sections is fundamental bottleneck to achieving high performance in multi-threaded programs.
- Locks do not always have to be acquired for a correct execution
- Paper proposes Speculative Lock Elision (SLE)
  - Dynamically remove unnecessary lock-induced serialization
  - Correct or repeat on misspeculation
  - No ISA updates required

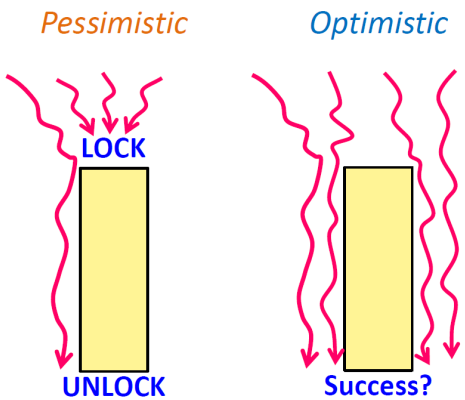


Figure: Elision of lock

Electrical & Computer ENGINEERING

Atomicity

For Guaranteeing atomicity, the following conditions must hold within critical section:

1. Data read within a speculatively executing critical section is not modified by another thread before the speculative critical section completes.
2. Data written within a speculatively executing critical section is not accessed (read or written) by another thread before the speculative critical section completes.

Assembly instructions have special constructs to implement this atomicity using Load-Linked (ldl_l) and Store Conditional (stl_c) instructions.

```
TEST, TEST & SET, UNSET

{   L1:1.ldl t0, 0(t1)    #t0 = lock
    2.bne t0, L1:         #if not free, goto L1
    3.ldl_l t0, 0(t1)     #load locked, t0 = lock
    4.bne t0, L1:         #if not free, goto L1
    5.lda t0, 1(0)        #t0 = 1
    6.stl_c t0, 0(t1)     #conditional store, lock = 1
    7.beq t0, L1:         #if stl_c failed, goto L1,
  8-15.<critical section>
{  16.stl 0 ,0(t1)        #lock = 0, release lock
```

Figure: Assembly code of LL/SC primitives

Electrical & Computer ENGINEERING

Algorithm

The complete algorithm for SLE is this:

1. If candidate load (ldl_l) to an address is followed by store (stl_c of the lock acquire) to same address, predict another store (lock release) will shortly follow, restoring the memory location value to the one prior to this store (stl_c of the lock acquire).
2. Predict memory operations in critical sections will occur atomically, and elide lock acquire.
3. Execute critical section speculatively and buffer results.
4. If hardware cannot provide atomicity, trigger misspeculation, recover and explicitly acquire lock if failed for *"restart threshold"* times
5. If second store (lock release) of step 1 seen, atomicity was not violated (else a misspeculation would have been triggered earlier). Elide lock-release store, commit state, and exit speculative critical section.

Electrical & Computer ENGINEERING

Implementation

Buffering Speculative State

1. ***Register State***
   - Reorder Buffer (ROB):
     1. Keep the speculative instructions here
     2. Limited by the size of ROB
   - Register Checkpoint:
     1. Sample before the speculative execution starts
     2. Imposes no constraint on the size of critical section

2. ***Memory State***
   - Speculative load is allowed in almost all processors
   - Speculative stores can be buffered in write buffer
     1. Size limitation

Electrical & Computer ENGINEERING

Implementation

Detecting Misspeculation

1. ***Atomicity Violation***
   - Reorder Buffer (ROB) used for SLE:
     1. No additional mechanism required
     2. The Load Store Queues are already snooped for external write
   - Register Checkpoint used for SLE:
     1. Add an access bit to mark element in cache as "accessed during critical execution"
     2. Works independent of cache levels
2. ***Resource Constraint***
   - Uncached access/events like system call
   - Finite Cache/write-buffer/ROB size
     1. Try to obtain the lock. If successful, commit the instructions

Electrical & Computer ENGINEERING

Results

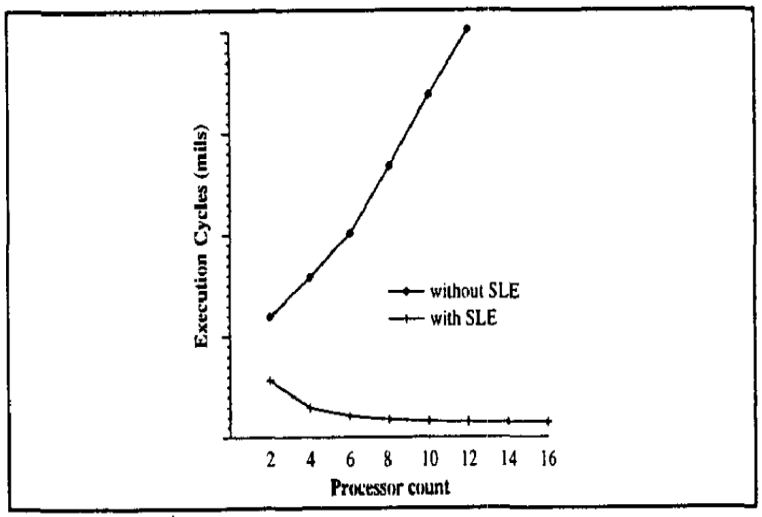


Figure: Microbenchmark result for CMP

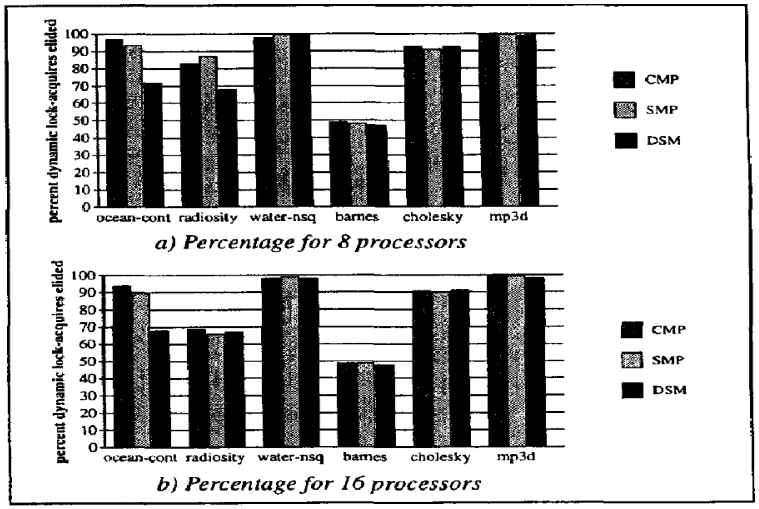


Figure: % of dynamic locks elided

Microbenchmark consists of N threads, each incrementing a unique counter $(2^{16})/N$ times, and all the N counters are protected by the same lock

A large fraction of Dynamic locks are elided. *Restart threshold* = 0, leads 10-30% fewer locks elided. ***Barnes*** has high contention for locked data.

Electrical & Computer ENGINEERING

Results

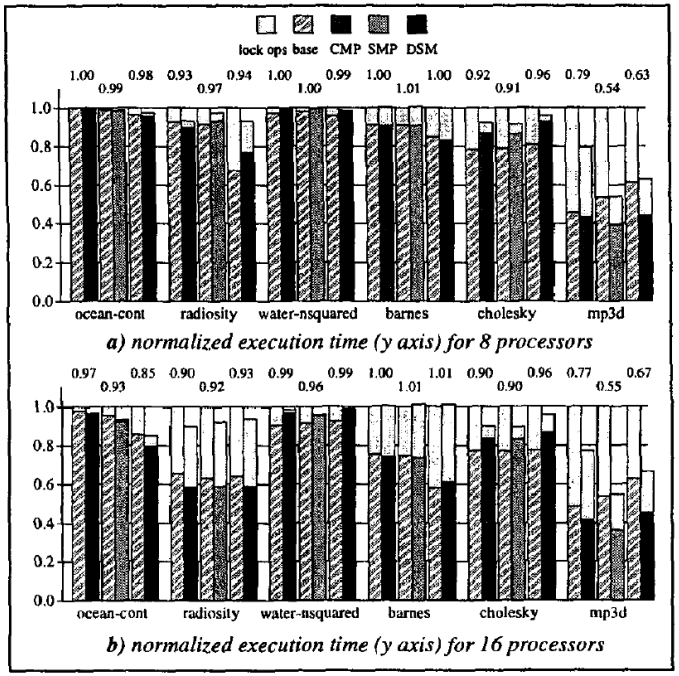


Figure: Normalized execution time (<1 means speedup)

Three major causes of speedup:

1. Concurrent execution
2. Reduced memory latency
3. Reduced memory traffic

Electrical & Computer ENGINEERING

Inferential Queuing and Speculative Push

Motivation and Contribution

- High communication miss rates in online transaction processing workloads, characterized by fine-grain updates of control data and frequent synchronization protecting such data
- These protected data migrates among processors with the passing of the lock and contribute to a large portion of the access latencies
- Processor stalls induced due to communication misses within critical sections will only increase over workloads
- Processors will be unable to generate misses early enough so as to hide memory access latencies to actively shared data.
- Two advancements can be done to speed-up the synchronization :
  - Lock requests can be queued and hence can be speculative forwarded to the immediate target
  - Using the queuing mechanism, if critical shared data can be forwarded along with the lock

Electrical & Computer ENGINEERING

IQL organization

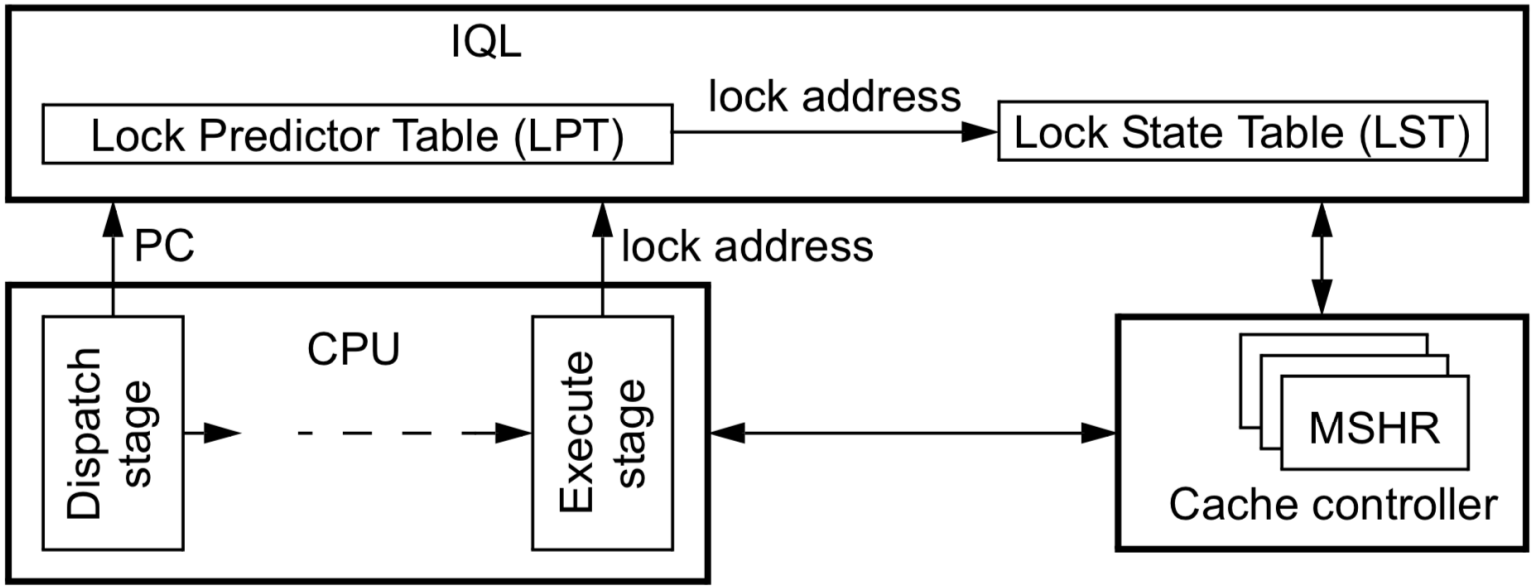


Figure: IQL Hardware organization

IQL implementation

- Read Exclusive Low-Priority (rd_X_lp)
  - A read for exclusive request, annotated with low priority (for locks)
  - Request can be deferred for a brief but bounded interval of time
- Lock Predictor Table (LPT)
  - Used for predicting the event of acquiring and releasing a lock by the processor
  - *The mechanism for these inferences is not discussed. What rate of mispredictions? Cost of mispredriction?*
- Lock State Table (LST)
  - Indexed by PC address of synchronizing instructions, identifying critical section
  - Tracks the state of locked line in the cache
  - Consulted on any incoming rd_X_lp request, if lock is HELD, request is buffered
- MSHR
  - Buffers rd_X_lp requests, to be services upon **inferred** release of lock

Electrical & Computer ENGINEERING

LST State Transitions : IQL implementation

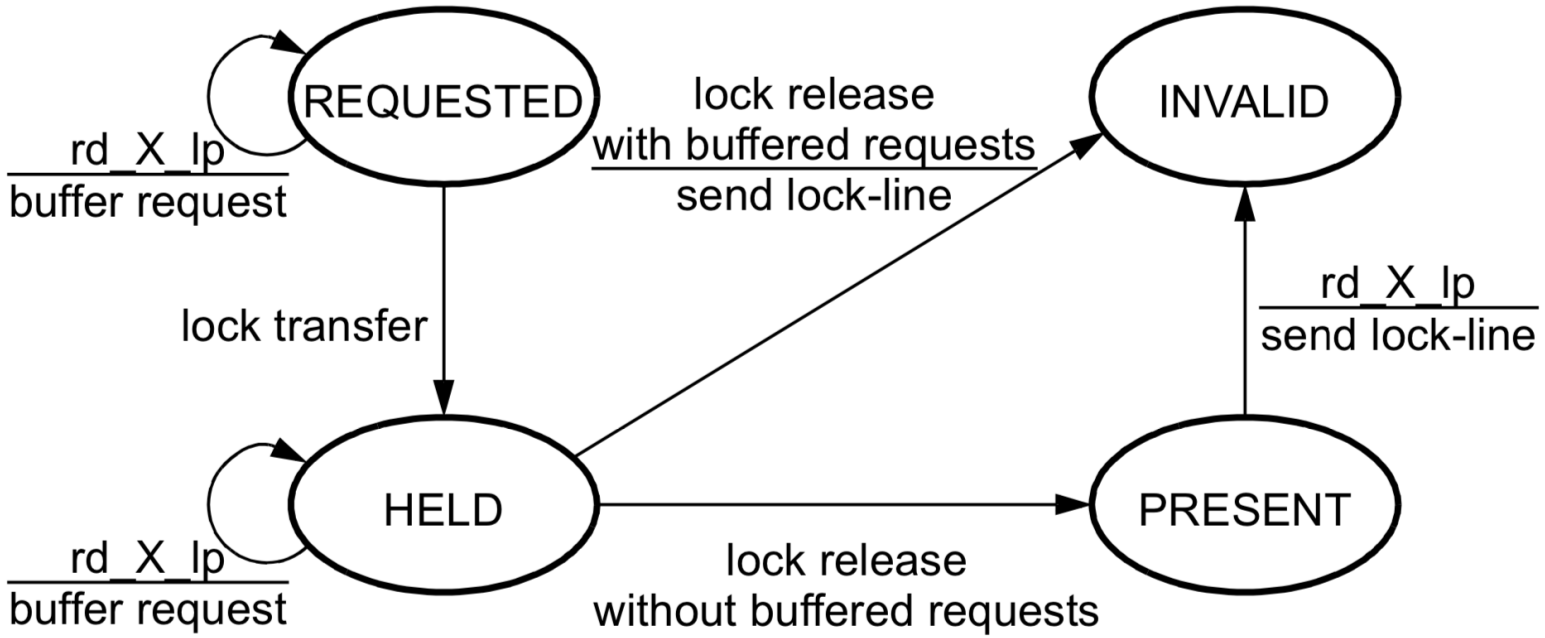


Figure: LST state transitions

- Incomplete state transition diagram, eg., there are no transitions out of INVALID
- State transition from PRESENT to HELD is required, to show necessity of PRESENT state

Electrical & Computer ENGINEERING

Speculative Push

LPT in IQL allows inference of the presence and extent of critical sections in programs. IQL also provides early knowledge of the next owner of lock. SP forwards the actively shared data to the next requesting processor, along with the lock.

- **Data Pairing** : Establish and record the association between critical section data and a lock
- **Prediction Confidence** : Enable/Disable the optimization by assigning a confidence level to pairings
- **Speculative Push** : Forwards any predicted data to the requesting processor along with the lock-line
  - Data transfer advantage : Initial access is overlapped with the lock transfer
  - Coherency Transfer advantage : It does not have to be upgraded for writing
  - Pushed data is also written back to the memory
  - Pushed cache line never evicts a valid cache line

Electrical & Computer ENGINEERING

SP Implementation : Data Pairing

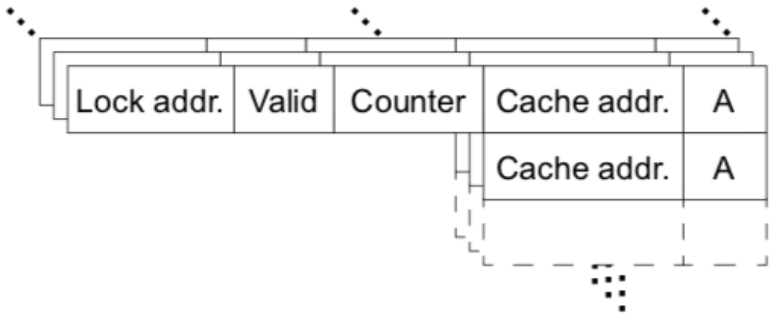
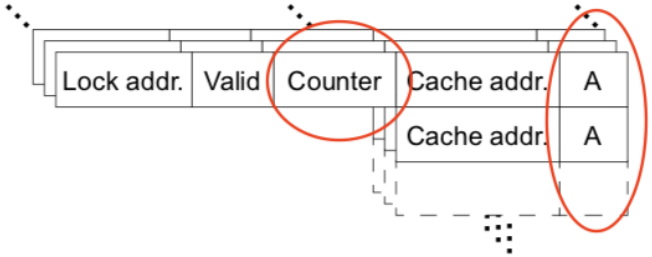


Figure: LST entry extended for Speculative Push

- Record addresses of accesses performed while the processor holds the lock
- These addresses becomes candidates for forwarding and are stored in LST along with the lock address
- Candidates for future pushes :
  - Write misses during a critical section
  - Lines that have been speculatively pushes into the cache (from previous lock acquire)

Electrical & Computer ENGINEERING

SP Implementation : Prediction Confidence



- A saturating counter is added to each LST entry data address to assign confidence in the data for forwarding
- Each cache address associated with the lock has an access-bit, which is used to track if these addresses were accessed during the lock
- The counter is set as per the number of addresses associated with it in the LST, accessed during the lock
- Maximum value : Enable optimization; Minimum value : Disable optimization
- Repeated evictions : Addresses accessed inside a critical section vary from one execution to the next preventing effective data forwarding
- *Can we send back feedback from next owner of lock to the processor who pushes? For selective prediction.*

Electrical & Computer ENGINEERING

SP Implementation : Forwarding Data Path

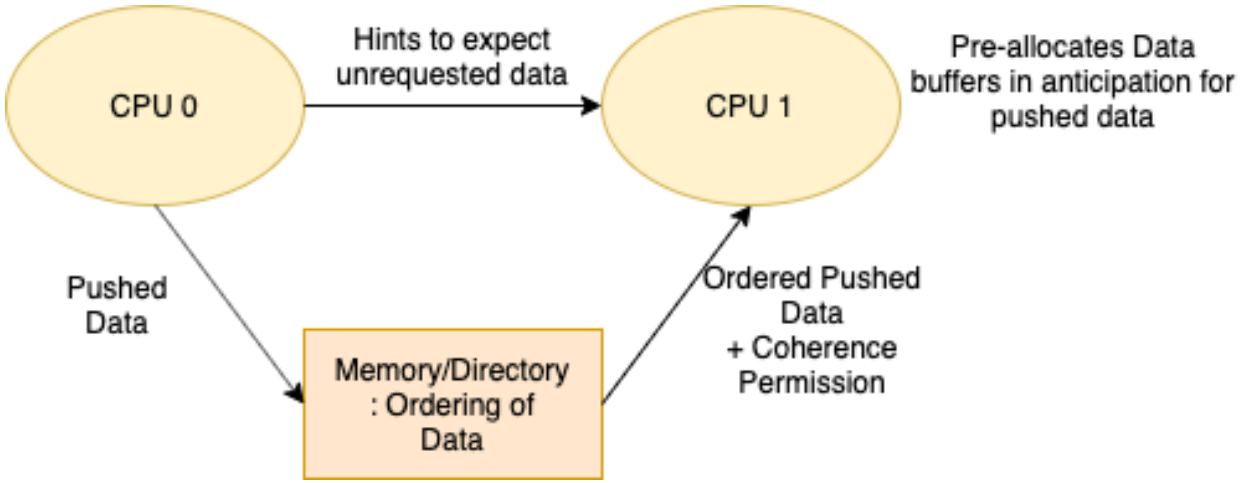


Figure: Data forwarding: Directory-based system

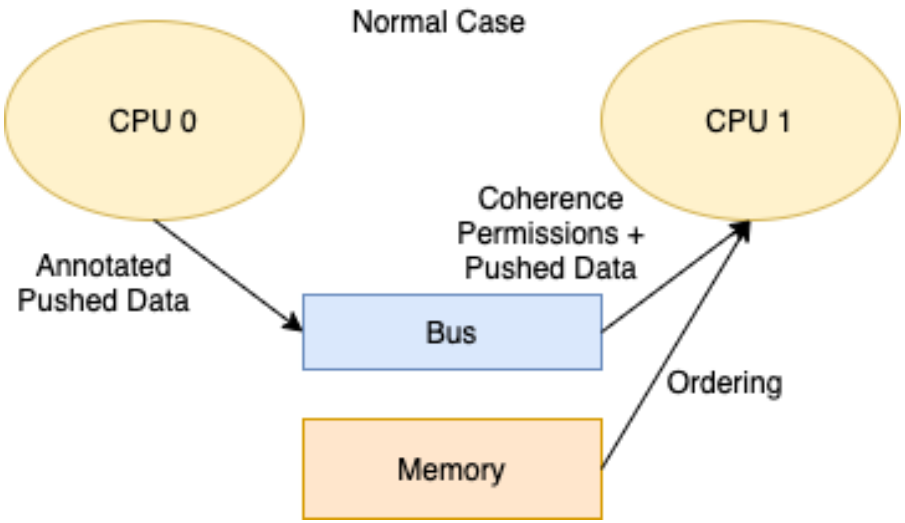


Figure: Data forwarding: Broadcast-based system

Electrical & Computer ENGINEERING

SP implementation : Ordering Speculative Push

Broadcast-based System

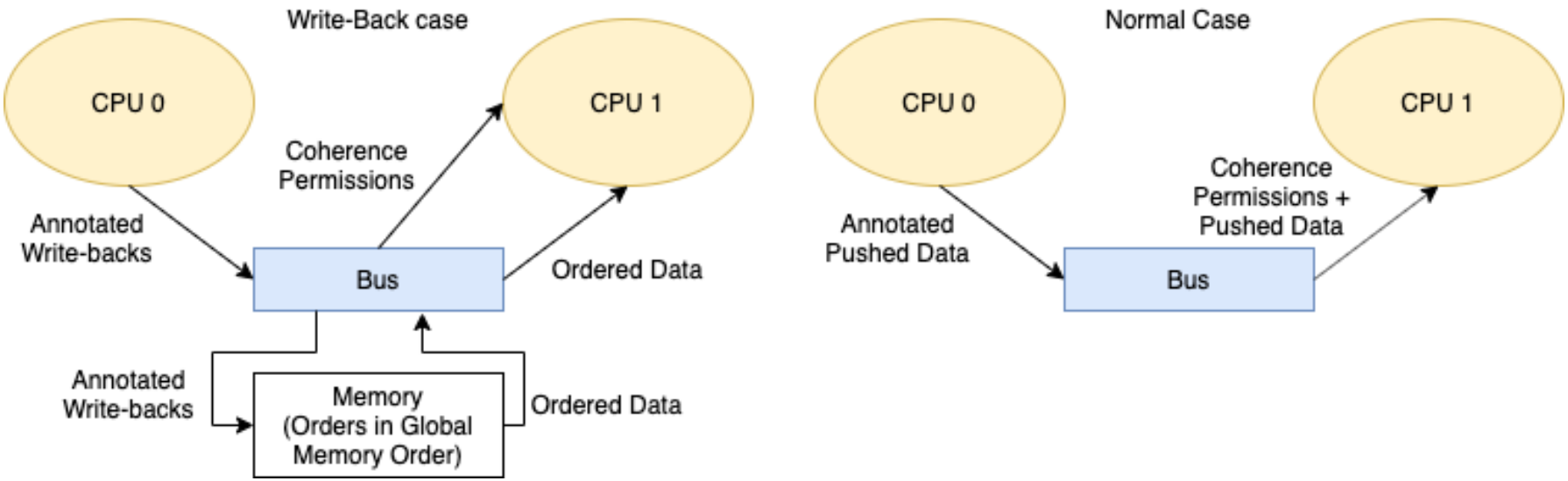


Figure: Data Ordering: Broadcast-based system

SP implementation : Ordering Speculative Push

Directory-based System

Responding to the annotated write-back, the directory node communicates with the target node, granting coherence permission (exclusive) or sending a NACK to the target node if necessary.

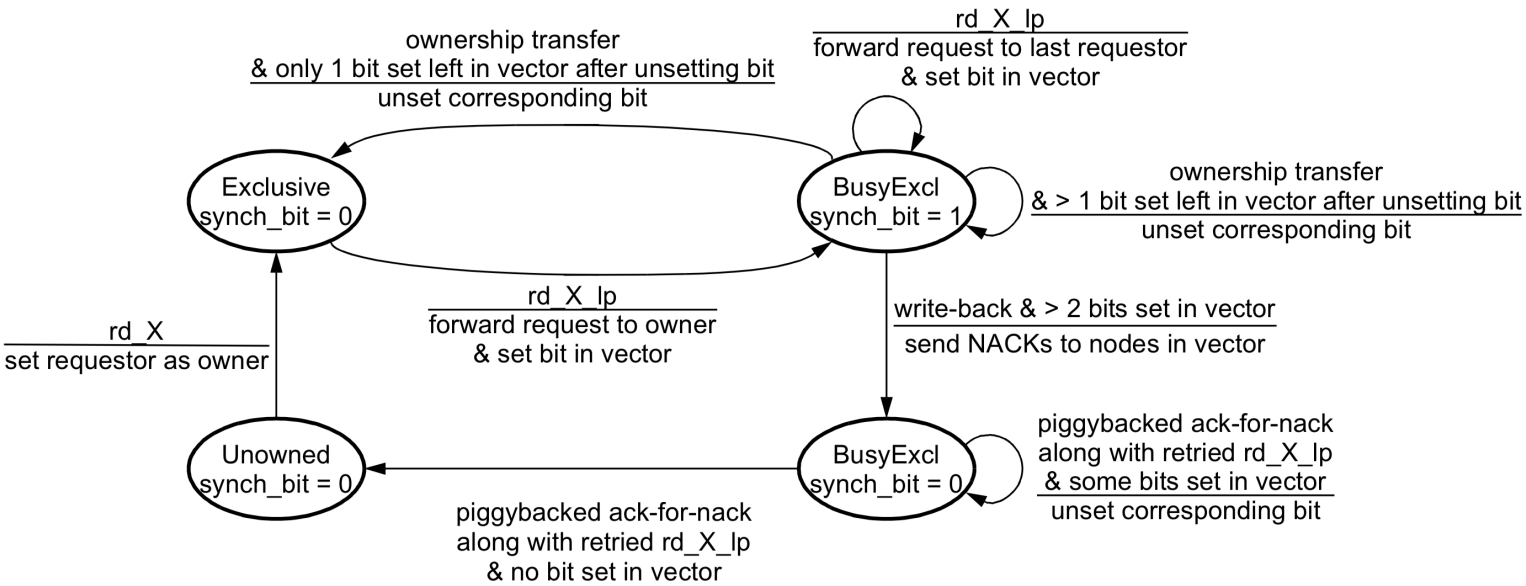


Figure: Protocol for IQLs: Directory-based system

Why is write-back really an exception? Since the lock will only be acquired by the next requestor, can it not fetch correct value from memory?

Electrical & Computer ENGINEERING

Matching Pushed Data with Coherence Permissions

- If a push is rejected, the corresponding coherence permission must also be rejected
- The push/coherence permission information is stored in a small table at the cache controller
- Both messages in the pair will occur exactly once, so every push received is tracked until its corresponding coherence permission is received, and vice versa
- An entry is removed when the pair is matched up
- Requestor includes in a rd_X_lp an indication of the number of lines it can track
- *This bookkeeping can become a bottleneck for performance, hardware for it is not discussed*
- *Why not send coherency permissions along with the data instead?*

Electrical & Computer ENGINEERING

Results

	Cholesky	MP3D	Raytrace	Water-Nsq
Without IQL (base)	(11)	(373)	(121)	(18)
IQL	1.13	1.21	2.75	1.04
IQL+SP	1.12	1.60	4.15	1.11

Figure: IQL+SP performance for a Directory-based system

	used		evicted		rejected		invalidated	
	SMP	DSM	SMP	DSM	SMP	DSM	SMP	DSM
Cholesky	8.00%	12.02%	92.00%	88.98%	0.00%	0.00%	0.00%	0.00%
MP3D	99.46%	99.56%	0.43%	0.40%	0.09%	0.04%	0.02%	0.00%
Raytrace	80.80%	71.64%	16.55%	28.33%	2.64%	0.00%	0.01%	0.03%
Water	93.80%	99.61%	6.20%	0.08%	0.00%	0.31%	0.00%	0.00%

Figure: SP prediction effectiveness

- ***cholesky*** - Little correlation exists between a lock address and data addresses accessed while the lock is held
- ***raytrace*** - Highly contended locks, large speedup in some cases (Not in SMP)
- ***Water-nsq*** - Communicates comparatively far more infrequently

Electrical & Computer ENGINEERING

Results

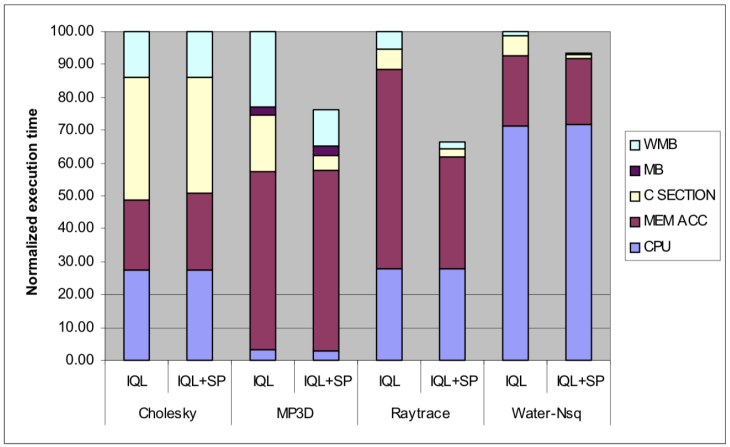


Figure: Stall Contributions | Directory-based system

	Cholesky	MP3D	Raytrace	Water-Nsq
IQL+Flush	1.00	1.13	1.28	1.04
IQL+SP	0.99	1.32	1.51	1.07

Figure: Comparison - SP versus Flush performance | Directory-based system

Flush : Flushing data back to the memory at the end of a critical section.
Avoids penalty of accessing remote dirty data, instead finds the desired data at memory directly.

Electrical & Computer ENGINEERING