Synchronization

- Enable a processor to reason about the temporal behavior of other processors
- Driving synchronization problem:
  - Critical sections: allow only one processor to perform a given task
- Other common constructs
  - Barriers
  - Producer-consumers
  - ...

- Hardware primitives, software libraries

Synchronization Primitives

- Atomic read-modify-write instructions
  - Examples:
    - swap

```
Acquirelock:
        lda     1,s0     /* create lock value */
        swap    s0,0(a0)/* swap register w/ lock*/
        bne     s0, Acquirelock
        ret
```

    - test & set (TAS)
    - fetch & increment
    - compare & swap (CAS)

- Problem: Key resources (memory, bus) tied down for the operation

Synchronization Primitives (2)

- Atomic pair of instructions
  - Load linked/store conditional
  - Interrupts, context switches, intervening writes to the same cache block cause store-conditional to fail
  - Link register tracks the lock address
  - Critical sections are limited to a few instructions, no memory operations

```
Acquirelock:
        lda     1,s0      /* create lock value */
        ldq_l   s1,0(a0) /* read lock linked */
        stq_c   s0,0(a0) /* set lock */
        bne     s1, Acquirelock
        ret
AtomicInc:
        ldq_l   s0,0(a0) /* read value linked */
        addq    s0,1,s0   /* increment */
        stq_c   s0,0(a0) /* set new value */
        ret
```

Synchronization APIs

- Build high-performance synchronization primitives in software on top of hardware primitives
  - Spin locks

```
Acquirelock:
        tas     s0,0(a0)/* test and set lock*/
        bne     s0, Acquirelock
        ret
```

- Synchronization operations are expensive
  - Test and test-and-set - spin in cache

```
Acquirelock:
        ldq     s1,0(a0) /* test */
        bne     s1, Acquirelock
        tas     s0,0(a0)/* test and set */
        bne     s0, Acquirelock
        ret
```

- Spin lock problems
  - wasted cycles on a non-dedicated processor - yield
  - competition when unlocked - exponential backoff
  - fairness - queuing

Synchronization APIs

- Other common constructs:
  - Barrier synchronization
    - Wait at a barrier until all processors have reached it
    - Suited for algorithms that operate in phases
  - Producer-consumer
    - Suited for pipelined algorithms
- New approach: optimistic concurrency
  - Allow all processors to perform potentially conflicting operations in parallel
  - Use a compare and swap to perform a transactional update
    - Read object, copy it, create a new object, replace with CAS
  - If CAS fails, retry
  - Pros: no deadlock, fault-tolerant
  - Cons: livelock, unfair to slow hosts, difficult to structure programs such that all dependencies are explicit in a single pointer

Choosing Primitives

- Are all primitives equally powerful ?
- Definitions:
  - Set consensus: assigning a set of numbers uniquely to N processors
  - Wait-free: an algorithm where every process finishes the protocol in a bounded number of steps, regardless of interleaving
  - Consensus N: Solves set consensus for N processors
- A synchronization primitive of consensus N cannot be used to implement other wait-free primitives >N.
  - 1-consensus:
    - memory read/write
  - 2-consensus:
    - Test&set, Fetch&Increment, Swap, Queue, Stack
  - Infinite-consensus:
    - Compare&Swap, LL/SC
- Within a given consensus number, some primitives are easier to implement
    - TAS vs. Swap
    - LL/SC vs. Compare&Swap

Synchronization on a Uniprocessor

- Execution on a traditional uniprocessor is sequential
  - BUT interrupts create asynchrony
- Disable context switches
  - Very common in OS code, simple, supports devices
  - Problems:
    - may delay interrupts, RT requirements
    - SPL cannot be entrusted to applications
- Restartable atomic sequences (RAS)
  - On every context switch, see if you interrupted an atomic operation.

```
Testandset:     /* NO SPECIAL INSTRUCTIONS */
        ldq     s0,0(a0) /* read lock value */
        bne     s0, tasfail /* test */
        orq     s0,1,s0   /* create lock value */
        stq     s0,0(a0)  /* set lock */
Interrupt: ...
        if(pc>=testandset && pc<(testandset+16))
                pc = testandset; /* rollback */
```

Example Hardware Primitives

- Alpha 21164
  - Load linked/store conditional
- MIPS
  - Load linked/store conditional
- Sparc
  - Compare & swap, test & set, etc.
- P6
  - Compare & swap, Compare & Swap 2