

Shared memory Synchronization

Note Title

3/3/2009

mutual exclusion

- how much in hardware?

speed vs. flexibility

sol'n is typically a form of atomic read-modify-write

- components of sync. event

acquire event
wait alg
release

- permission to proceed

Waiting algs

- block

deschedule to OS

- busy-wait

spin wait on variable

- hybrid

Cray xmp - lock registers

Blue Gene done from IBM - low latency ICW

read-modify-write inst. are common

simple software lock

lock:	ld	reg, loc	/# copy loc to reg #/
	cmp	reg, #0	
	bnz	lock	/# if not 0, try again #/
	st	loc, #1	/# mark as locked #/
	ret		
unlock:	st	loc, #0	/# mark unlocked #/
	ret		

atomic exchange inst.

specifies loc. & reg.

value in loc. \rightarrow reg.

another val (maybe func. of value read) \rightarrow loc.

simple example test & set

(value in loc. \rightarrow reg.
const 1 \rightarrow loc.

Success if value loaded to reg is 0

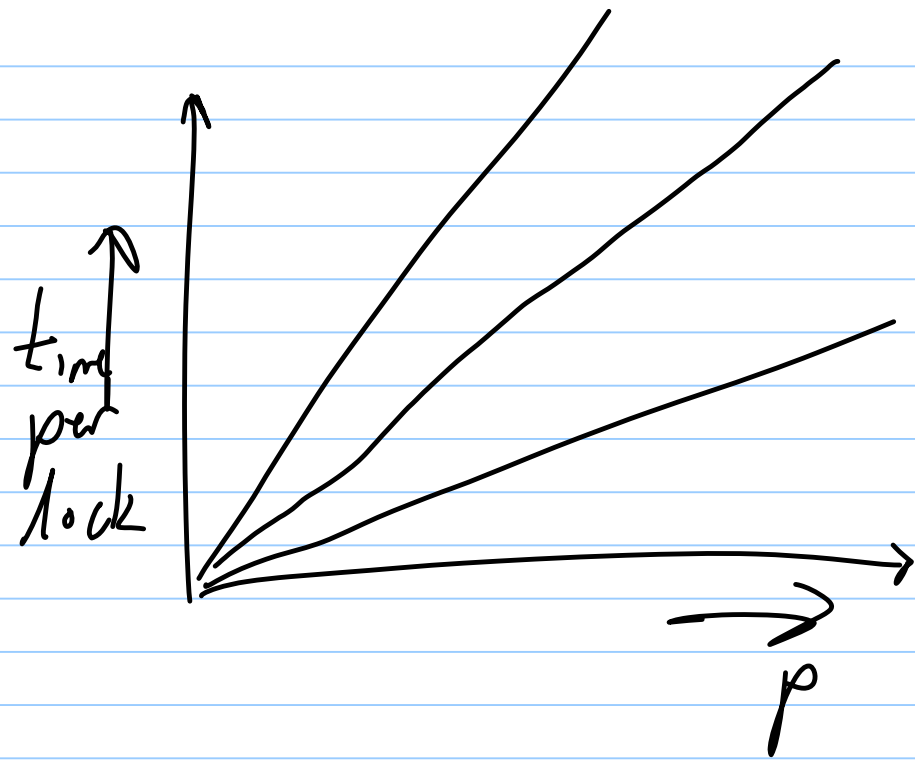
test & set lock

lock: t & s reg, loc.
 bnz lock // if not 0, try again
 ret

unlock st loc, #0
 ret

Other atomic inst.

Swap
Fetch & op
Compare & Swap



loop
lock
delay (d)
unlock
end

reduce freq. of issuing tds while waiting
busy-wait with reads (loads)

test and tds

dimensions of performance

low uncontested latency
traffic with contention

storage
fairness

diff ISA

- fast w/ reads

- failed r-m.w attempts to not generate inval. det. w

- flexibility

load locked (-linked), store conditional

LL

SC

- LL reads variable \rightarrow reg. from loc.
- arb. inst stream
- SC tries to store back to loc. iff
no proc. has written to loc. since this proc. LL
if SC succeeds, 3 steps were atomic
if " fails, need to retry LL
succ indicated by cond. codes

lock using ll/sc

```
lock:  ll  reg1, loc
      sc  loc, reg2
      <cond branch> lock
      ret
```

```
unlock: st  loc, #0
      ret
```

ticket lock array-based queueing lock