Queue Locks and Local Spinning

Some Slides based on:
The Art of Multiprocessor Programming
by Maurice Herlihy & Nir Shavit
Memory Models

- Memory Contention
- Communication Contention
- Communication Latency

Cache Coherent (CC) vs. Distributed Shared Memory (DSM)
Today: Revisit Mutual Exclusion

• Think of performance, not just correctness and progress
• Begin to understand how performance depends on our software properly utilizing the multiprocessor machine’s hardware
Remote Access

• Remote access is expensive!

• Allow spinning only on local variables:
  – DSM: spin only on variables in the local memory
  – CC: spin only on variables in cache
Basic Spin-Lock

Spin lock

Critical section

Resets lock upon exit
Basic Spin-Lock

...lock suffers from contention - no local spinning!
Idea

• Avoid useless invalidations
  - By keeping a queue of threads
• Each thread
  - Notifies next in line
  - Without bothering the others
Anderson Queue Lock

flags

acquired

acquiring

getAndIncrement

next

T
F
F
F
F
F
F
F
F
F
F
F
Anderson Queue Lock

• Good
  - Local spinning (CC model)
  - Simple, easy to implement

• Bad
  - One bit per thread
    • Unknown number of threads?
    • Small number of actual contenders?
CLH Lock

- FIFO order
- Small, constant-size overhead per thread
Initially

idle

tail → false
Green Wants the Lock

acquiring

tail → false
Green Wants the Lock

acquiring

tail

false → true
Green Wants the Lock

acquiring

Swap

tail

false

true
Green Has the Lock

acquired

tail

false

true
Blue Wants the Lock

acquired

acquiring

false

true

tail

true

Art of Multiprocessor Programming
Blue Wants the Lock

acquired

acquiring

Swap
Blue Wants the Lock

acquired

acquiring

false

true

true

tail

Art of Multiprocessor Programming
Blue Wants the Lock

acquired

false

tail

acquiring

true

true
Blue Wants the Lock

acquired

acquiring

false

true

true

Implicitely Linked list

tail
Blue Wants the Lock

acquired

acquiring

false

true

true

Art of Multiprocessor Programming
Blue Wants the Lock

acquired

acquiring

Actually, it spins on cached copy

Art of Multiprocessor Programming
Green Releases

release  acquiring
false  false  true

Bingo!

Art of Multiprocessor Programming
Green Releases

released

acquired

tail

true

Art of Multiprocessor Programming
CLH Queue Lock

• Entry section

```plaintext
new myNode
myNode := true
do myPred := tail while !CAS(tail,myPred,myNode)
wait until !myPred
```

• Exit section

```plaintext
myNode := false
```
CLH Lock

• Good
  - Lock release affects predecessor only
  - Small, constant-sized space

• Bad
  - Not local spinning for DSM model
CLH Lock

- Each thread spin's on predecessor's memory
- Could be far away ...
MCS Lock

- FIFO order
- Spin on local memory only
- Small, Constant-size overhead
Initially

idle

false
Acquiring

(allocate Qnode)

tail

false

true
Acquiring

acquiring

swap

tail

false

true
Acquiring
Acquiring

acquired

acquiring

false

true

tail

swap

Art of Multiprocessor Programming 34
Acquiring
Acquiring

acquired

acquiring

false

tail

true
Acquiring

acquired

acquiring

false

true

tail

Art of Multiprocessor Programming 37
Acquiring
Acquiring

Acquired

True

False

Yes!
MCS Queue Lock

• Entry section

```python
new myNode
do myPred := tail while !CAS(tail, myPred, myNode)
If myPred!=null
    myNode.locked:= true
    myPred.next:= myNode
    wait until !(myPred.locked)
```

• Exit section

```python
If myNode.next == null
    if CAS(tail, myNode, null) then return
    wait until myNode.next!=null
    myNode.next.locked := false
```
Green Release

releasing

swap

false

false
By looking at the queue, I see another thread is active.
Green Release

By looking at the queue, I see another thread is active.

I have to wait for that thread to finish.
Green Release

releasing

prepare to spin

false   true

Art of Multiprocessor Programming
Green Release

releasing

spinning

false

true
Green Release

releasing

spinning

false

false
Green Release

releasing Acquired lock

false

false

false
Non-Uniform Memory Architecture (NUMA)
Non-Uniform Memory Architecture (NUMA)

• Today, many large scale modern multiprocessors are NUMA:
  - Clusters of processors with shared local memory
  - Access by a processor to the memory of its cluster two or more times faster than remote memory
  - Per cluster cache
Lock Bouncing
Hierarchical Locks

• Encourage threads with high mutual memory locality to acquire the lock consecutively
• Reduce overall cache misses
Hierarchical CLH (HCLH) Lock

[Luchangco, Nussbaum and Shavit 2006]

• Local queue per cluster
• Global queue to enter the critical section
• A local queue is added to the global queue with a single CAS
HCLH Lock

• First, add the thread to the local queue
• If a thread is the first in the local queue, it is responsible for merging into the global queue
HCLH Lock

acquiring

Local tail

false
HCLH Lock

Local tail

acquiring

Successor_must_wait

Tail_when_merged

false

cid  true  false
HCLH Lock

acquiring

Local tail

Swap

Successor_must_wait

Tail_when_merged

false

cid true false
HCLH Lock

acquiring

Local tail

false

cid true false
HCLH Lock

Local tail

acquiring

false

acquiring

cid  true  false

cid  true  false
HCLH Lock

acquiring

Local tail

Swap

cid true false

cid true false

false

true

false
HCLH Lock

Local tail

acquiring false

acquiring cid true false

cid true false
HCLH Lock

acquiring

Local tail

false

cid true false
HCLH Lock

Local tail

```
cid  true  false
```
HCLH Lock

Cluster master: sees lock is held, so waits a “combining delay”
HCLH Lock

Local tail

Cluster master: sees lock is held, so waits a “combining delay”

Global tail
HCLH Lock

Local tail

Global tail

SWAP
HCLH Lock

Local tail

Global tail
HCLH Lock
References
