Efficient Synchronization: Let Them Eat QOLB

Matthew Moskewicz

CS258, UC Berkeley, 2002.04.19

Scope of Work

- Fine grained parallel shared memory programs running on distributed shared memory cache coherent multiprocessors.
  - Bam.
- Locks and Barriers are the one true method of explicit synchronization.
  - But Barriers are uninteresting.
  - Message passing? Nope.
- So, this work is all about locks.

Breaking down the Lock

- We want to break down the time spent dealing with locks, from the cosmic perspective.
- Proposed breakdown of synch period into three phases: (all for one lock)
  - Transfer
    - Time from: A release complete → B acquire complete
  - Load/Compute
    - Time from: B acquire complete → B compute complete
  - Release
    - Time from: B compute complete → B release complete

Their illustrative figure:

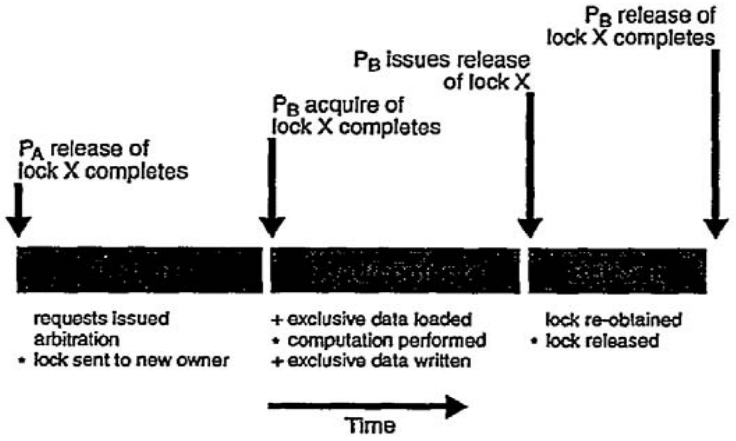

Figure 1 Breakdown of one synchronization period.

Optimization Frontier

- Local spinning
  - Reduces network load
- Queue based locking
  - No arbitration, quicker transfer
- Collocation
  - Transfer data with locks
- Synchronous Prefetch
  - Get lock/data in advance

	Phase of the synchronization period				
	Transfer		Load/compute		Release
Synchronization mechanisms	Arbitration	Lock transfer	Data read	Data write	Re-obtain lock
Local spinning					
Queue-based locking	✓	✓			✓
Collocation			✓	✓	
Synchronous prefetch		✓	maybe		

Table 1 How synchronization mechanisms reduce overhead.

Please don't upset the primitives

- Good 'ol Test and Set (TS)
  - And his buddy, Test and Test and Set (TTS)
- The MCS lock
  - And his uppity cousins, the LH and M locks
  - Queue based locking primitives
- Reactive synchronization
  - Watch level of contention, adjust lock type
  - TS for low contention, MCS for high
- QOLB
  - The queen of all locks. All hail QOLB.
  - Just hardware MCS? But apparently not quite.

Variants

- Exponential back off
  - Applies to TS, TTS, does about what you'd think.
- Collocation
  - Applies to all primitives (not used on LH, M, R(?))
  - Transfer data with lock
- Prefetching
  - Applies to all primitives (only used with QOLB)

Simulation Environment

- WWT
  - Okay, sounds fine in general
  - Fully connected constant delay p-p network? What the?
    - But I guess it's okay 'cause they try real hard to explain why it's okay.
  - 32 Processors, CC-NUMA, SCI CCP
    - There they go with that SCI thing again.
  - Release consistent
    - Use two implementations: SC and 'a more aggressive one' which doesn't say too much. But they add a confusing detail or two.

Microbenchmark

- Everybody grab the (one) lock, quick!
- Shows effect of contention, kills TS, TTS
  - TS+E, TTS+E better, but still suck
  - Queue locks are good (somebody's always got it, but some queuing overhead unavoidable)
  - Queue locks are even better if you magically set overhead to near 0. (QOLB)

Microbenchmark Graph

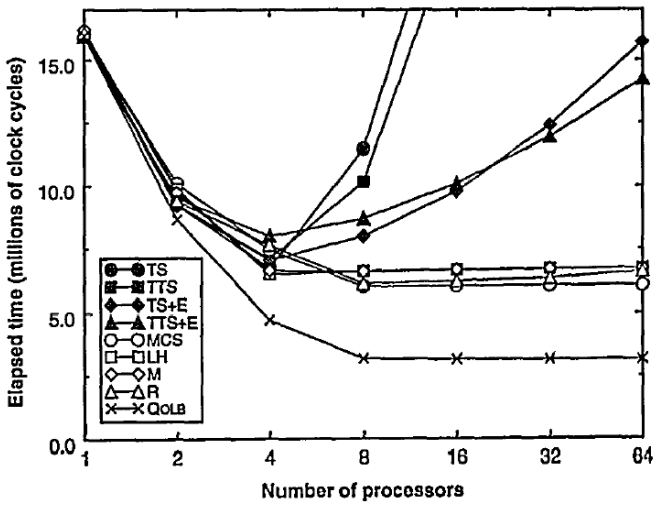

Figure 3 **Microbenchmark performance.**

Marcobenchmark Results

	Benchmark									
	Barnes		Mp3d		Ocean		Pthor		Raytrace	
Experiment	SEQ	REL	SEQ	REL	SEQ	REL	SEQ	REL	SEQ	REL
TS	(190)	0.94	(231)	1.02	(16.5)	1.19	(221)	1.16	(826)	1.22
TS+C	1.67	1.85	1.03	1.12	1.31	1.69	0.86	1.13	2.47	2.56
TS+E	1.17	1.40	0.86	1.21	1.12	1.37	0.88	1.22	2.06	2.15
TS+E+C	1.31	1.67	0.90	1.29	1.31	1.68	0.93	1.34	2.56	2.65
TTS	1.02	1.11	1.05	1.11	1.02	1.22	1.04	1.23	1.03	1.12
TTS+C	1.72	1.87	1.09	1.18	1.32	1.70	0.95	1.36	2.54	2.61
TTS+E	1.17	1.40	0.83	1.18	1.11	1.40	0.87	1.21	2.03	2.15
TTS+E+C	1.32	1.66	0.87	1.25	1.26	1.70	0.94	1.35	2.56	2.65
MCS	1.57	1.61	1.18	1.30	1.24	1.55	1.06	1.25	2.31	2.26
MCS+C	1.58	1.63	1.25	1.36	1.25	1.65	1.17	1.37	2.29	2.33
LH	1.21	1.48	0.81	1.12	1.24	1.55	0.87	1.22	2.26	2.31
M	1.21	1.47	0.75	1.06	1.24	1.55	0.87	1.18	2.25	2.29
R	1.19	1.47	0.76	1.08	1.19	1.49	0.87	1.20	2.28	2.35
QOLB	1.79	1.83	1.46	1.60	1.31	1.65	1.11	1.34	2.58	2.64
QOLB+C	1.89	1.92	1.65	1.75	1.34	1.70	1.25	1.51	2.62	2.69
QOLB+C+P	1.89	1.92	1.65	1.75	1.31	1.68	1.26	1.54	2.63	2.70
QOLB+C+CP	1.89	1.93	1.64	1.74	1.35	1.70	1.25	1.53	2.63	2.70

Table 5 **Speedups of different synchronization primitives.** The numbers in parentheses represent the execution time (in millions of clock cycles) for the particular benchmark running on sequentially consistent hardware. The other numbers represent speedups, calculated as the ratio of the execution time of the base run to that of the optimized synchronization primitive.

Macrobenchmark Discussion

- Unsurprising the QOLB wins, given methodology
- But TTS+C does almost as well, save mp3d
  - And QOLB basically just wins because it assumes lower overhead due to extra hardware, and mp3d exploits this (one assumes)
  - But so what? It still wins, so add the hardware, right? It's easy, right?
    - Probably not. Easy only wrt SCI …
    - And one app is less than convincing

Low cost QOLB?

- Single microbenchmark, dubious result
  - Winner is CQL, unless you add +C to QOLB
    - But, uh, why didn't we add +C to CQL again?

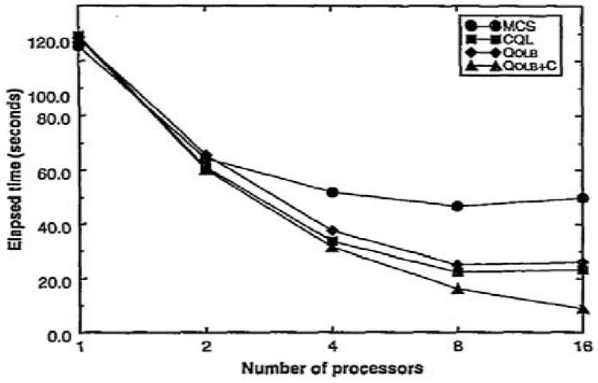

Figure 5 Performance of software QOLB.

Summary

- If you compare the same operation in software to a faster hardware version, the faster hardware version is faster.
- I'd need to see (much) more impressive results to justify complex hardware locks.
- I'd especially want to see modified applications, message passing, sockets, and so on.