Comparing Locks

EVERYTHING YOU ALWAYS WANTED TO KNOW ABOUT SYNCHRONIZATION BUT WERE AFRAID TO ASK
TUDOR DAVID, RACHID GUERRAOUI, VASILEIOS TRIGONAKI

PHILIP TAFFET
Outline

Motivation
Locks
Target Platforms
Performance Comparison Results
Some Conclusions
Applications (SSYNC)
Motivation

There are many types of locks and even more types of architectures. Does locking performance depend on architecture? How? Which lock is best?
Locks

Test and set (TAS)
Test and test and set (TTAS)
Ticket lock
Array-based lock
MCS lock
CLH lock
Hierarchical CLH lock (HCLH)
Hierarchical Ticket lock (HTICKET)

FIFO Locks
Locks

Test and set (TAS)
Test and test and set (TTAS)
Ticket lock
Array-based lock
MCS lock
CLH lock
Hierarchical CLH lock (HCLH)
Hierarchical Ticket lock (HTICKET)
Hierarchical CLH

Local CLH queue per cluster (socket)

One global queue

Qnode at the head of the global queue holds the lock

A Hierarchical CLH Queue Lock; Victor Luchangco, Dan Nussbaum, Nir Shavit
Acquire lock - Step 1: Enqueue locally

Socket 1

A
B
A
B
C

Tail pointer

Socket 2

1
2

Tail pointer

Tail pointer
Acquire lock - Step 2: Spin wait

Socket 1

Socket 2

Tail pointer
Acquire lock - Step 3: Combining delay
Acquire lock- Step 4: Splicing
Acquire lock- Step 5: Spin wait
Hierarchical Ticket

Similar concept with two levels of locking

When a thread acquires the local lock, it attempts to acquire the global lock

Global lock not released until the local queue is empty
Systems

Opteron
- Directory based cache coherence protocol.
- Directory located in LLC.

Xeon
- Broadcast snooping

Niagara
- Coherence via shared L2 cache on other side of chip

Tilera
- Coherence via shared L2 cache distributed across chip
Opteron

Average dist: 1.25 hops
Xeon

Average distance:
1.375 hops

= 1 socket = 10 cores
Niagara

<table>
<thead>
<tr>
<th>8-way MT</th>
<th>L1</th>
<th></th>
<th>L2 Cache</th>
</tr>
</thead>
<tbody>
<tr>
<td>8-way MT</td>
<td>L1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8-way MT</td>
<td>L1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8-way MT</td>
<td>L1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8-way MT</td>
<td>L1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8-way MT</td>
<td>L1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8-way MT</td>
<td>L1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8-way MT</td>
<td>L1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8-way MT</td>
<td>L1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Niagara: A 32-way Multithreaded Sparc Processor; Poonacha Kongetira, Kathirgamar Aingaran, Kunle Olukotun
Comparing Lock Types

Higher is better

Conclusion: There is no universally best lock.
Why the variations?

<table>
<thead>
<tr>
<th>System</th>
<th>Opteron (2.1 GHz)</th>
<th>Xeon (2.13 GHz)</th>
<th>Niagara (1.2 GHz)</th>
<th>Tilera (1.2 GHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hops</td>
<td>same die</td>
<td>same MCM</td>
<td>one hop</td>
<td>two hops</td>
</tr>
<tr>
<td>Modified</td>
<td>81</td>
<td>161</td>
<td>172</td>
<td>252</td>
</tr>
<tr>
<td>Owned</td>
<td>83</td>
<td>163</td>
<td>175</td>
<td>254</td>
</tr>
<tr>
<td>Exclusive</td>
<td>83</td>
<td>163</td>
<td>175</td>
<td>253</td>
</tr>
<tr>
<td>Shared</td>
<td>83</td>
<td>164</td>
<td>176</td>
<td>254</td>
</tr>
<tr>
<td>Invalid</td>
<td>136</td>
<td>237</td>
<td>247</td>
<td>327</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>System</th>
<th>Operation loads</th>
<th>Operation stores</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modified</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Owned</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Exclusive</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Shared</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Invalid</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Operation</td>
<td>all</td>
<td>all</td>
</tr>
<tr>
<td>Modified</td>
<td>110</td>
<td>197</td>
</tr>
<tr>
<td>Shared</td>
<td>272</td>
<td>283</td>
</tr>
</tbody>
</table>

atomic operations: Compare & Swap (C), Fetch & Increment (F), Test & Set (T), Swap (S)
Why the variations?

Relative performance of atomic primitives and cache operations varies widely in the hardware.
⇒ varying performance of locks
Effect of contention

Amount of contention affects performance of the lock
  ◦ E.g. test-and-set is good in a low-contention situation but bad in a high-contention situation

Should the programmer have to predict how much contention there will be for a given lock?
Main observations

Crossing sockets is killer (2x to 7.5x worse performance vs. intra-socket)

It’s hard to avoid cross-socket communication (OS scheduler, incomplete cache directory, etc.)

Loads, stores can be as expensive as atomic operations (Non-local access can be a bottleneck)

Intra-socket non-uniformity matters (Hierarchical locks scale better on non-uniform systems)

Consider message passing for highly contended data (Message passing may be faster)

There’s no universally best lock (Pick a lock based on architecture and expected contention)

Simple locks are powerful (Ticket lock performs best in many cases)
Main observations

- **Crossing sockets is killer** (2x to 7.5x worse performance vs. intra-socket)
- **It’s hard to avoid cross-socket communication** (OS scheduler, incomplete cache directory, etc.)
- **Loads, stores can be as expensive as atomic operations** (Non-local access can be a bottleneck)
- **Intra-socket non-uniformity matters** (Hierarchical locks scale better on non-uniform systems)
- **Consider message passing for highly contended data** (Message passing may be faster)
- **There’s no universally best lock** (Pick a lock based on architecture and expected contention)
- **Simple locks are powerful** (Ticket lock performs best in many cases)
  - Except when they aren’t
Conclusions

Queueing locks deliver good performance on most platforms, especially under high contention.

Standard system-dependent library with optimized lock implementations for that platform?

Because low-overhead is so important (Simple locks are powerful), it’s hard to try anything fancy without creating excessive overhead.

Idea: Adaptive locks that change type based on contention they detect

  http://doi.acm.org/10.1145/1837853.1693489