Locks on Multicore and Multisocket Platforms

John Mellor-Crummey

Department of Computer Science
Rice University

johnmc@rice.edu
Context

• Last lecture: locks and barriers
  • Lock synchronization on multicore platforms
• Upcoming
  — transactional memory
  — practical non-blocking concurrent objects
Papers for Today

• **Everything you always wanted to know about synchronization but were afraid to ask.** David Tudor, Rachid Guerraoui, and Vasileios Trigonakis. In Proceedings of SOSP '13. ACM, New York, NY, USA, 33-48.

Motivation for Studying Lock Performance

• There are many types of locks and architectures
• Does lock 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)
- Hierarchical backoff lock
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 backoff lock
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 backoff lock

Locks we haven’t discussed
Why Hierarchical Locks?

NUMA architectures

• Not all memory is equidistant to all cores
  —each socket has its own co-located memory
  —consequence of scaling memory bandwidth with processor count

• Today’s systems: system-wide cache coherence

• Access latency depends on the distance between the core and data location
  —memory or cache in local socket
  —memory or cache in remote socket

• Multiple levels of locality
  —0 hop, 1 hop, 2 hop, ...
Locks on NUMA Architectures

• Problem:
  — passing locks between threads on different sockets can be costly
  — overhead from passing lock and data it protects
  — data that has been accessed on a remote socket produces long latency cache misses

• Solution:
  — design locks to improve locality of reference
  — encourage threads with mutual locality to acquire a given lock consecutively

• Benefits:
  — reduce migration of locks between NUMA nodes
  — reduce cache misses for data accessed in a critical section
Hierarchical CLH

• Structure
  — local CLH queue per cluster (socket)
  — one global queue
  — qnode at the head of the global queue holds the lock

• Operation: when a node arrives in the local queue ...
  — delay for a bit to let successors arrive
  — move a batch from a socket queue to the global queue
    - CAS local tail into global tail
    - link local head behind previous global tail

Hierarchical CLH in Action

Socket J

L_J

Socket K

L_K

has lock; in critical section

spin

spin
Hierarchical CLH in Action

Socket J

Socket K

pause

spin

spin

has lock; in critical section
Hierarchical CLH in Action

- G
  - J
    - has lock; in critical section
    - spin
  - K
    - spin
  - L

- Socket J
  - LJ
    - pause
    - spin

- Socket K
  - LK
Hierarchical CLH in Action

Socket J

Socket K

G has lock; in critical section

spin

pause

spin

L_J

L_K

pause
Hierarchical CLH in Action

G

Socket J

L_J

Socket K

L_K

has lock; in critical section

pause spin spin

pause spin
Hierarchical CLH in Action

- G
- Socket J
- Socket K

- L_J
- L_K

- has lock; in critical section
- linking
- spin
- spin
- pause
- spin

16
Hierarchical CLH in Action

Socket J

Socket K

G

has lock; in critical section

spin

spin

linking

spin

spin

L_J

L_K

pause

spin
Hierarchical CLH in Action

Socket J

Socket K

has lock; in critical section

pause

spin
Hierarchical CLH in Action

Socket J

L_J

Socket K

L_K

has lock; in critical section

spin

spin

spin

spin

spin

linking

spin
Hierarchical CLH in Action

has lock; in critical section

Socket J

Socket K

pause

linking

spin
Hierarchical CLH in Action

- Socket J
  - L_J
  - pause

- Socket K
  - L_K

G

spin

has lock; in critical section

spin

spin

spin

spin

spin

linking

spin
Hierarchical CLH in Action

Socket J

Socket K

pause

has lock; in critical section

spin

spin

spin

spin

spin

spin

spin

spin
Hierarchical Ticket

- Two levels of ticket locks
  - global
  - local: one per socket

- Two-level ticket lock (cohorting version by Dice et al.)
  - acquire
    - acquire local ticket
    - if flag “global granted” is set, proceed
    - else acquire global ticket lock
  - release
    - if successors available in local lock, set “global granted” for local lock and increment local ticket
    - otherwise, clear “global granted” for local lock and increment global ticket

- “Everything…” paper used a more complex version
  - https://github.com/tudordavid/libslock/blob/master/src/htlock.c
Hierarchical Backoff Lock

- Test-and-test-and-set lock with back off scheme to reduce cross node contention of a lock variable

- Use thread locality to tune backoff delay
  - when acquiring a lock
    - assign thread ID to lock state
  - when spin waiting
    - compare thread ID with lock holder and back off proportionally

- Limitations:
  - reduce lock migration only probabilistically
  - lots of invalidation traffic: costly for NUMA

Systems with Different Characteristics

- Opteron: 4 x AMD Opteron 6172 (48 cores)
  - directory based cache coherence
  - directory located in LLC
- Xeon: 8 x Intel Xeon E7-8867L (80 cores; SMT disabled)
  - broadcast snooping
- Niagara: SUN UltraSPARC-T2 (8 cores; 64 threads)
  - coherence via shared L2 cache on far side of chip
- Tilera: TILE-Gx CPU (36 cores)
  - coherence via distributed, shared L2 cache
Opteron Platform

- Opteron: 4 x AMD Opteron 6172 (48 cores)
- Each chip contains two 6-core dies
- MOESI protocol, directory based cache coherence
  —directory located in LLC
- Average distance: 1.25 hops

Figure credit: Everything you always wanted to know about synchronization but were afraid to ask. D. Tudor, R. Guerraoui, and V. Trigonakis. In Proceedings of SOSP '13. ACM, New York, NY, USA, 33-48.
Xeon Platform

- Xeon: 8 x Intel Xeon E7-8867L (80 cores; SMT disabled)
  — broadcast snooping
- 10 cores per socket
- Average distance: 1.375 hops

Figure credit: Everything you always wanted to know about synchronization but were afraid to ask. D. Tudor, R. Guerraoui, and V. Trigonakis. In Proceedings of SOSP '13. ACM, New York, NY, USA, 33-48.
Niagara

- **Niagara: SUN UltraSPARC-T2** (8 cores; 64 threads)
  — coherence via shared L2 cache on far side of chip

Figure credit: Niagara: A 32-way Multithreaded SPARC Processor; P. Kongetira, K. Aingaran, K. Olukotun
Tilera

- Tilera: TILE-Gx CPU (36 cores)
  - coherence via distributed, shared L2 cache

Figure credit: http://www.tilera.com/sites/default/files/productbriefs/TILE-Gx8036_PB033-02_web.pdf
## Operation Latency Across Platforms

Latencies depend upon distance and (sometimes) state

<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></td>
<td>same die</td>
<td>same MCM</td>
<td>same die</td>
<td>same core</td>
</tr>
<tr>
<td>Hops</td>
<td>one hop</td>
<td>two hops</td>
<td>one hop</td>
<td>other core</td>
</tr>
<tr>
<td></td>
<td>loads</td>
<td></td>
<td></td>
<td></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>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>stores</td>
<td></td>
</tr>
<tr>
<td>Modified</td>
<td>83</td>
<td>172</td>
<td>191</td>
<td>273</td>
</tr>
<tr>
<td>Owned</td>
<td>244</td>
<td>255</td>
<td>286</td>
<td>291</td>
</tr>
<tr>
<td>Exclusive</td>
<td>83</td>
<td>171</td>
<td>191</td>
<td>271</td>
</tr>
<tr>
<td>Shared</td>
<td>246</td>
<td>255</td>
<td>286</td>
<td>296</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>atomic operations: Compare &amp; Swap (C), Fetch &amp; Increment (F), Test &amp; Set (T), Swap (S)</td>
<td></td>
</tr>
<tr>
<td>Operation</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Modified</td>
<td>Shared</td>
<td></td>
<td></td>
</tr>
<tr>
<td>all</td>
<td>120</td>
<td>324</td>
<td>430</td>
<td></td>
</tr>
<tr>
<td>all</td>
<td>120</td>
<td>324</td>
<td>430</td>
<td></td>
</tr>
<tr>
<td>all</td>
<td>71</td>
<td>108</td>
<td>64</td>
<td>95</td>
</tr>
<tr>
<td>all</td>
<td>71</td>
<td>108</td>
<td>64</td>
<td>95</td>
</tr>
<tr>
<td>all</td>
<td>77</td>
<td>51</td>
<td>70</td>
<td>63</td>
</tr>
<tr>
<td>all</td>
<td>77</td>
<td>51</td>
<td>70</td>
<td>63</td>
</tr>
<tr>
<td>all</td>
<td>124</td>
<td>82</td>
<td>121</td>
<td>95</td>
</tr>
<tr>
<td>all</td>
<td>124</td>
<td>82</td>
<td>121</td>
<td>95</td>
</tr>
<tr>
<td>all</td>
<td>142</td>
<td>102</td>
<td>141</td>
<td>115</td>
</tr>
<tr>
<td>all</td>
<td>142</td>
<td>102</td>
<td>141</td>
<td>115</td>
</tr>
</tbody>
</table>

**Opteron:** load latency independent of state

**Xeon:** load latency depends on state

Figure credit: Everything you always wanted to know about synchronization but were afraid to ask. D. Tudor, R. Guerraoui, and V. Trigonakis. In Proceedings of SOSP '13. ACM, New York, NY, USA, 33-48.
Variation in Performance of Atomics

Throughput: Higher is better

Observations

—relative performance of atomic primitives and cache operations varies widely in the hardware

—varying performance of locks is in part due to varying performance of atomic operations
Lock Performance vs. Platform

Observations
— throughput on multi-socket systems is lower than on single chips
— there is no universally best lock

Throughput: Higher is better
Lock Acquisition vs. Previous Owner

Figure credit: Everything you always wanted to know about synchronization but were afraid to ask. D. Tudor, R. Guerraoui, and V. Trigonakis. In Proceedings of SOSP ’13. ACM, New York, NY, USA, 33-48.
Impact of Contention on Performance

Figure 7: Throughput of different lock algorithms using 512 locks.

Figure 5: Throughput of different lock algorithms using a single lock.
• Crossing sockets is expensive
  — 2x to 7.5x slower than intra-socket
  — hard to avoid cross-socket communication
    — e.g., Opteron: incomplete cache directory (no sharer info)

• Loads, stores can be as expensive as atomic operations
  — non-local access can be a bottleneck

• Intra-socket non-uniformity matters (e.g., Tilera vs. Niagara)
  — hierarchical locks scale better on non-uniform systems

• Simple locks can be effective
  — ticket lock performs best in many cases

• There’s no universally optimal lock
  — optimal lock depends upon architecture and expected contention
Impact of Contention on Performance

Figure 7: Throughput of different lock algorithms using 512 locks.

Figure 5: Throughput of different lock algorithms using a single lock.
Lock Cohorting
Lock Cohorting

- Idea: use two levels of locks
  - global locks
  - local locks, one for each socket or cluster (NUMA node)
- First in socket to acquire local lock
  - acquire socket lock then the global lock
  - pass local lock to other waiters in the local node
  - eventually relinquish global lock to give other nodes a chance
- Recipe for NUMA-aware locks without special algorithms
- Cohorting can compose any kind of lock into a NUMA lock
  - augments properties of cohorted locks with locality preservation
- Benefits
  - reduces average overhead of lock acquisition
  - reduces interconnect traffic for lock and protected data
Global and Local Locks for Cohorting

- **Global lock G**
  - thread-oblivious: acquiring thread can differ from releasing thread
  - globally available to all nodes of the system

- **Local lock S**
  - supports cohort detection
    - a releasing thread can detect if other threads waiting
  - records last state of release as global or local

- Once S is acquired
  - local release → proceed to critical section
  - global release → try to acquire G

- Upon release of S
  - IF may_pass_local OR alone? → release globally
  - ELSE → release locally
Global and Local Locks for Cohorting

- C-BO-BO lock
  - Global backoff (BO) lock and local backoff locks per node
- C-TKT-TKT lock
  - Global ticket lock and local ticket (TKT) locks per node
- C-BO-MCS lock
  - Global backoff lock and local MCS lock
- C-MCS-MCS lock
- C-TKT-MCS lock

- Use of abortable locks in cohort designs needs extra features to limit aborting while in a cohort
  - A-C-BO-BO lock
  - A-C-BO-CLH lock (queue lock of Craig, Landin, & Hagersten)
Experiments

- Microbenchmark LBench is used as a representative workload
- LBench launches identical threads
- Each thread loops as follows
  - acquire central lock
  - access shared data in critical section
  - release lock
  - ~4ms of non-critical work
- Run on Oracle T5440 series machine
  - 256 hardware threads
  - 4 NUMA clusters
- Evaluation shows that cohort locks outperform previous locks by at least 60%
Average Throughput vs. # of Threads

Throughput/sec

# of Threads

MCS
HBO
HCLH
FC-MCS
C-BO-BO
C-TKT-TKT
C-BO-MCS
C-TKT-MCS
C-MCS-MCS
Conclusions: Cohorting is Useful

• Useful design methodology
  — no special locks required
  — can be extended to additional levels of locality
    - e.g., tile based systems where locality is based on grid position
    - multiple levels of lock cohorts

• Cohort locks improve performance over previous NUMA aware lock designs

• Performance scaling with thread count is better with locality-preserving cohort locks
New Work: Adaptive HMCS Lock

- Tree of MCS locks to exploit multiple levels of locality
- Fast path: directly acquire root if lock is available
- Hysteresis: adaptively select at which level to compete
Performance of AHMCS on Power 4-Socket

Throughput: Higher is better

(a) 1 cache line
(b) 2 cache lines
(c) 4 cache lines
(d) 64 cache lines