Everything You Always Wanted to Know About Synchronization but Were Afraid to Ask

Tudor David, Rachid Guerraoui and Vasileios Trigonakis

Ecole Polytechnique Federale de Lausanne (EPFL)
Multi-Core

- Multi-core is used in many systems
- Then number of core $\uparrow$, Performance $\uparrow$? NO

Synchronization is one of the biggest scalability bottlenecks
Synchronization

• Why does we use?
  ▪ Concurrent access to shared data
  ▪ To ensure the orderly execution

• Why is synchronization bottleneck?
  ▪ Hardware
  ▪ Synchronization algorithm
  ▪ Application context
  ▪ Workload
Cache Coherence

• Multi-core system have a separate cache for each core
  ▪ Write operation break consistency among caches

• Cache coherence
  ▪ To maintain caches of a common memory resource
Cache Coherence protocols

• MSI protocol
Cache Coherence Protocols

• MESI protocol
  ▪ Added exclusive state
    – No other has a copy of this cache line
  ▪ Reduced expensive invalidate operation

• MOESI protocol
  ▪ Added owned state
    – This cache line has been modified but there might be more shared copy on other core
  ▪ Reduced expensive write operation to memory
Cache Coherence Example

• Acquiring lock process

Acq(lock);

Processor
Cache
State
Inval
Data
Held=1

Update
Invalidate

Acq(lock);

Processor
Cache
State
Mod
Data
Held=1

Read-Exclusive

Shared memory (held = 1)
What to deal with

• Hardware Processors
  ▪ Multi-sockets
    – AMD Opteron
      • 4 x 6172 – 48 cores
    – Intel Xeon
      • 8 x E7-8867L – 80 cores
  ▪ Single-sockets
    – Sun Niagara 2
      • 8 cores
    – Tilera TILE-Gx36
      • 36 cores

• Synchronization layer
  ▪ Concurrent software
    – Hash table, etc.
  ▪ Primitives
    – Lock, etc.
  ▪ Atomic operations
    – Compare & swap, etc.
  ▪ Cache coherence
    – Load & store
Hardware-Level Analysis
Local Accesses

**Opteron**

- Within socket: 40 ns

**Zeon**

- Within socket: 20 – 40 ns
Remote Accesses

**Opteron**
- Within socket: 40 ns
- Per hop: +40 ns

**Zeon**
- Within socket: 20 – 40 ns
- Per hop: +50 ns
### Operation Latency – Multi Socket

<table>
<thead>
<tr>
<th>System State</th>
<th>Hops</th>
<th>Opteron</th>
<th>Xeon</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>same</td>
<td>same</td>
</tr>
<tr>
<td></td>
<td></td>
<td>die</td>
<td>MCM</td>
</tr>
<tr>
<td></td>
<td></td>
<td>one hop</td>
<td>two</td>
</tr>
<tr>
<td></td>
<td></td>
<td>hops</td>
<td>hops</td>
</tr>
<tr>
<td>Loads</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Modified</td>
<td>81</td>
<td>161</td>
<td>252</td>
</tr>
<tr>
<td>Owned</td>
<td>83</td>
<td>163</td>
<td>254</td>
</tr>
<tr>
<td>Exclusive</td>
<td>83</td>
<td>163</td>
<td>253</td>
</tr>
<tr>
<td>Shared</td>
<td>83</td>
<td>164</td>
<td>254</td>
</tr>
<tr>
<td>Invalid</td>
<td>136</td>
<td>237</td>
<td>327</td>
</tr>
<tr>
<td>Stores</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Modified</td>
<td>83</td>
<td>172</td>
<td>273</td>
</tr>
<tr>
<td>Owned</td>
<td>244</td>
<td>255</td>
<td>291</td>
</tr>
<tr>
<td>Exclusive</td>
<td>83</td>
<td>171</td>
<td>271</td>
</tr>
<tr>
<td>Shared</td>
<td>246</td>
<td>255</td>
<td>296</td>
</tr>
</tbody>
</table>

Crossing sockets is a killer
Up to 7.5x more expensive
Single-Socket Processors

Niagara

• Equidistant from the cache
• Uniform: 23ns

Tilera

• Non uniform
• 1 hop: 40ns
• Per hop: +2 ns
## Operation Latency – Single Socket

<table>
<thead>
<tr>
<th>System State</th>
<th>Niagara</th>
<th>Tilera</th>
<th>T</th>
<th>M</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Hops</td>
<td>same core</td>
<td>other core</td>
<td>one hop</td>
</tr>
<tr>
<td>Modified</td>
<td>3</td>
<td>24</td>
<td>45</td>
<td>65</td>
</tr>
<tr>
<td>Owned</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Exclusive</td>
<td>3</td>
<td>24</td>
<td>45</td>
<td>65</td>
</tr>
<tr>
<td>Shared</td>
<td>3</td>
<td>24</td>
<td>45</td>
<td>65</td>
</tr>
<tr>
<td>Invalid</td>
<td>176</td>
<td>176</td>
<td>118</td>
<td>162</td>
</tr>
</tbody>
</table>

### Loads

<table>
<thead>
<tr>
<th>System State</th>
<th>Niagara</th>
<th>Tilera</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modified</td>
<td>24</td>
<td>57</td>
</tr>
<tr>
<td>Owned</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Exclusive</td>
<td>24</td>
<td>57</td>
</tr>
<tr>
<td>Shared</td>
<td>24</td>
<td>86</td>
</tr>
</tbody>
</table>

Uniform is expected to scale better, The non-uniform is affected both distance and the number of involved cores.
Atomic Operations – Multi Sockets

- Very fast single-thread performance
  - But drops on two or more cores and decreases further when there is cross-socket communication
Atomic Operations – Single Sockets

• Lower single-thread throughput
  ▪ But scale to a maximum value
Software-Level Analysis
Analysis Scope

• 9 Locks
  ▪ Spinlocks
    – Test and test-and-set lock (TTAS), Ticket lock
  ▪ Queue based lock
    – Array based lock, CLH lock, MCS lock
  ▪ Hierarchical lock
    – Hierarchical CLH lock, Hierarchical ticket lock
  ▪ Mutex

• Concurrent software
  ▪ Hash table
Ticket Lock

Acquired Ticket : 0

Acquiring Ticket : 1

Acquiring Ticket : 2

Acquiring Ticket : 3

Acquiring Ticket : 4

Lock

Next ticket : 1

Now serving : 0

Spin

Spin

Spin

Spin
Ticket Lock

Release

Acquiring Ticket: 1

Acquiring Ticket: 2

Acquiring Ticket: 3

Acquiring Ticket: 4

Next ticket: 2
Now serving: 1
CLH Lock

Acquiring

tail \rightarrow false
CLH Lock

Acquired

tail → true → false

prev
reference
CLH Lock

Acquiring

Spin

tail
false
true
false

prev
reference

Acquired
CLH Lock

Acquiring
Spin

Unlock

false
false
false

prev
reference
CLH Lock

Acquired

prev
reference

tail
true
false
false
Hierarchical Lock
Hierarchical Lock

- NUMA aware lock
  - Using local cache for lock
Locks Microbenchmark

- Initialize $N$ locks & $T$ threads
- Each thread repeatedly
  - Chooses one lock out of $N$ at random
  - Acquires the lock
  - Reads and writes the protected data
  - Releases the lock
- Repeat with 9 different lock algorithms
  - spinlocks, queue-based, hierarchical, mutex
- Report the best total throughput
Locks on Multi Sockets

High contention (4 locks)  
Low contention (128 locks)

Multi sockets provide limited scalability due to higher latencies of remote access

X:Y, X: the scalability over the single-thread execution  
Y: the best-performance lock
Locks on Single Sockets

High contention (4 locks)
Low contention (128 locks)

Complex locks are generally the best under extreme contention,
Simple locks perform better under low contention

X:Y, X: the scalability over the single-thread execution
Y: the best-performance lock
Hash Table – best locks

Simple locks are powerful

25 / 32
Conclusion

• Crossing sockets is a killer
  ▪ Up to 7.5x more expensive communication

• Intra-socket uniformity matters

• Simple locks are powerful
  ▪ Better in 25 out of 32 data-points on a hash table