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

Tudor David, Rachid Guerraoui, Vasileios Trigoniakis
The multi-core revolution

- Big challenges in hardware & software
- Software
  - **scalability**: ↑ performance by ↑ number of cores

Synchronization is one of the biggest scalability bottlenecks
Synchronization

• Cannot always be avoided
  – not all applications embarrassingly parallel

• Synchronization is just an overhead
  – but guarantees correctness

• Scalability of synchronization
  – do not ↓ performance as the number of cores ↑

Scalability of synchronization is key to application scalability
Synchronization is difficult

• Tons of work
  – design of synchronization schemes [ISCA’89, TPDS’90, ASPLOS’91, TOCS’91, PPoPP’01, PODC’95, ICPP’06, SPAA’10, IPDPS’11, PPoPP’12, ATC’12, …]
  – fix synchronization bottlenecks [SOSP’89, HPCA’07, OSDI’99, OSDI’08, SOSP’09, APLOS’09, OSR’09, OSDI’10, …]

• Scalability issues?
  – hardware
  – usage of specific atomic operations
  – synchronization algorithm
  – application context
  – workload

Limited understanding of the behavior of synchronization
Take a step back and perform a thorough analysis of synchronization on modern hardware.

What is the main source of scalability problems in synchronization?

**Answer**

Scalability of synchronization is mainly a property of the hardware.
Key observations

1. Crossing sockets is a killer

2. Sharing within a socket is necessary but not sufficient

3. Intra-socket uniformity matters

4. Loads & stores can be as expensive as atomic operations

5. Simple locks are powerful

...
Disclaimer 😊

We do not claim
“Bad synchronization” in software will scale well due to hardware

We claim
“Good synchronization” in software might not scale as expected due to hardware
Analysis method

Hardware processors

Multi-sockets
• AMD Opteron (4x 6172 - 48 cores)
• Intel Xeon (8x E7-8867L - 80 cores)

Single-sockets
• Sun Niagara 2 (8 cores)
• Tilera TILE-Gx36 (36 cores)

Synchronization layers

software primitives (e.g., locks)
atomic operations (e.g., CAS)
cache coherence (load and store)
Outline

1. Crossing sockets
2. Sharing within a socket
3. Intra-socket uniformity
4. Atomic operations
5. Simple locks

applications
primitives
atomic ops
cache coherence
Distance on multi-sockets

Opteron

- Within socket: 40 ns
- Per hop: +40 ns
- Up to \(3\times\) more

Xeon

- Within socket: 20 – 40 ns
- Per hop: +50 ns
- Up to \(8\times\) more

Crossing sockets is a killer: up to \(8\times\) more expensive
Locks on multi-sockets

** Each point is the best result out of 9 lock algorithms

### High contention (4 locks)

**Locks microbenchmark**

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

### Low contention (512 locks)
Locks on multi-sockets

** Each point is the best result out of 9 lock algorithms

**High contention** (4 locks)

**Low contention** (512 locks)

Crossing sockets is a killer: big decrease in performance

11/4/2013

Vasileios Trigonakis (EPFL)
Hash table on multi-sockets

** Each point is the best result taken by any out of 9 lock algorithms

### High contention (12 buckets)

<table>
<thead>
<tr>
<th>Threads</th>
<th>Opteron</th>
<th>Xeon</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 socket</td>
<td>15</td>
<td>?</td>
</tr>
<tr>
<td>18</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>36</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

### Low contention (512 buckets)

<table>
<thead>
<tr>
<th>Threads</th>
<th>Opteron</th>
<th>Xeon</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 socket</td>
<td>10</td>
<td>?</td>
</tr>
<tr>
<td>18</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>36</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

Crossing sockets is a killer
Outline

1. Crossing sockets
2. Sharing within a socket
3. Intra-socket uniformity
4. Atomic operations
5. Simple locks
Coherence on multi-sockets

Opteron

Incomplete directory

Xeon

Broadcast requests
Locality on multi-sockets

**Opteron**
- **Within socket**: 40 ns
- Data within a socket
  - served locally (40 ns)
  - broadcast (120 ns)

**Xeon**
- **Within socket**: 20 – 40 ns
- Data within a socket
  - served by the LLC (20 – 40 ns)

Sharing within a socket is necessary but not sufficient.
Locks on multi-sockets

High contention (4 locks)

Throughput (Mops/s)

Low contention (512 locks)

Sharing within a socket is not sufficient
Hash table on multi-sockets

High contention (12 buckets)

Low contention (512 buckets)

Sharing within a socket is necessary but not sufficient
Outline

1. Crossing sockets
2. Sharing within a socket
3. Intra-socket uniformity
4. Atomic operations
5. Simple locks

applications
primitives
atomic ops
cache coherence
Distance on single-sockets

**Niagara**

- Uniform: 23 ns

**Tilera**

- 1 hop: 40 ns
- Per hop: +2 ns
- Up to 0.5x more

Uniformity is expected to scale better
Locks on single-sockets

High contention (4 locks)

Throughput (Mops/s)

<table>
<thead>
<tr>
<th>Threads</th>
<th>Niagara</th>
<th>Tilera</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3.8x</td>
<td>2.3x</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
<td></td>
</tr>
<tr>
<td>36</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Low contention (512 locks)

Throughput (Mops/s)

<table>
<thead>
<tr>
<th>Threads</th>
<th>Niagara</th>
<th>Tilera</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>25.0x</td>
<td>21.2x</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
<td></td>
</tr>
<tr>
<td>36</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Uniformity leads to up to 70% higher scalability
Hash table on single-sockets

High contention (12 buckets)

Low contention (512 buckets)

Uniformity leads to up to 50% higher scalability
Outline

1. Crossing sockets
2. Sharing within a socket
3. Intra-socket uniformity
4. Atomic operations
5. Simple locks
Atomic ops on local data

CAS: an order of magnitude more expensive on local data
Atomic ops on multi-sockets

Loads and stores can be as expensive as atomic operations
Outline

1. Crossing sockets
2. Sharing within a socket
3. Intra-socket uniformity
4. Atomic operations
5. Simple locks
Hash table – best locks

High contention (12 buckets)

Low contention (512 buckets)

Throughput (Mops/s)

Threads

Simple locks are powerful

11/4/2013
Lessons learned

1. **Crossing sockets is a killer**
   → up to 8x more expensive communication

2. **Sharing within a socket is necessary but not sufficient**
   → up to 3x more expensive communication

3. **Intra-socket uniformity matters**
   → up to 70% higher scalability

4. **Loads & stores can be as expensive as atomic operations**
   → 8 - 35% more expensive on non-locally cached data

5. **Simple locks are powerful**
   → better in 25 out of 32 data-points on a hash table

Scalability of synchronization is mainly a property of the hardware
Analysis’ space & limitations

- locks
- message passing
- lock-free
- combiner approaches
...

hardware platforms

synchronization schemes

AMD  Intel  Oracle  TILERA  IBM  ARM
SSYNC synchronization suite

- Systems / applications
  - Software primitives (e.g., locks)
  - Atomic operations (e.g., compare-and-swap)
  - Cache coherence (load and store)

- ssht, TM2C, Memcached
- libslock, libssmp
- ccbench

http://go.epfl.ch/ssync
1. Crossing sockets is a killer
2. Sharing within a socket is necessary but not sufficient
3. Intra-socket uniformity matters
4. Loads & stores can be as expensive as atomic operations
5. Simple locks are powerful

Scalability of synchronization is mainly a property of the hardware

http://go.epfl.ch/ssync