The Performance of Spin Lock Alternatives for Shared-Memory Multiprocessors

written by T. E. Anderson
1. Introduction
2. Multiprocessor Architectures
3. Simple approaches to spin lock
4. Software alternatives
5. Queueing Approach
6. Measurement Results
7. Hardware solution
8. Conclusion
1. INTRODUCTION

In 1990, many shared-memory multiprocessors architectures had been designed.

When one or more processors access shared-memory, it requires mutual exclusion.

Because pure software mutual exclusion is expensive, all H/D support mutual exclusion.
1. INTRODUCTION

Simple operation

Encapsulate into Single atomic instruction

More than one instruction, define critical section

Expected wait is Less than the cost of blocking/resuming

<table>
<thead>
<tr>
<th>Spin Lock</th>
<th>Blocking Lock</th>
</tr>
</thead>
</table>

This paper focuses on spin locks

GOAL:
1. providing scalability
2. low-latency spin-lock on multiprocessors
2. MULTIPROCESSORS ARCHI...

Two dimensions
- Interconnection (BUS, Multistage)
- Cache Coherence

Architectures considered in this paper
- Multistage with no CC
- Multistage with invalidation-based CC
- Bus with no CC
- Bus with snoopy write-through invalidation-based CC
- Bus with snoopy write-back invalidation-based CC
- Bus with snoopy distributed-write CC
Interconnection - Multistage

2 stages
2. MULTIPROCESSORS ARCHITECTURE

Interconnection - BUS

Diagram showing three CPUs connected to two memory (MEM) nodes through a bus interconnection.
Cache coherence Strategies

- CPU
  - Cache

INTERCONNECTION (Bus, Multistage)

Main Memory
2. MULTIPROCESSORS ARCHITECTURE

No Cache coherence

INTERCONNECTION (Bus, multistage)
Write-through Cache coherence

```
X = 1 valid
```

```
CPU1
    | Cache
    |
```

```
CPU2
    | Cache
    |
```

```
INTERCONNECTION (Bus, Multistage)
```

```
Main Memory
```

```
X = 1
```
Write-through Cache coherence

CPU1

Cache

CPU2

Cache

INTERCONNECTION (Bus, Multistage)

Main Memory

X = 2 valid

X = 1 valid
X = 1 invalid

X = 2
2. MULTIPROCESSORS ARCHITECTURE...

Write-through Cache coherence

CPU1 → Cache → Main Memory

CPU2 → Cache → Main Memory

X = 2 valid

X = 1 invalid

X = 2 valid

INTERCONNECTION (Bus, Multistage)
2. MULTIPROCESSORS ARCHI...

Write-Back Cache coherence

CPU1

Cache

CPU2

Cache

INTERCONNECTION (Bus, Multistage)

Main Memory

X = 1 Exclusive

X = 1
Write-Back Cache coherence

- CPU1 connects to Cache
- CPU2 connects to Cache
- INTERCONNECTION (Bus, Multistage)
- Main Memory

X = 2 Modified
X = 1
2. MULTIPROCESSORS ARCHITECTURE...

Write-through Cache coherence

CPU1

Cache

X = 2 Shared

INTERCONNECTION (Bus, Multistage)

CPU2

Cache

X = 2 Shared

Main Memory

X = 2
2. MULTIPROCESSORS ARCHI...

Distributed Write Cache coherence

CPU1

X = 1

Cache

CPU2

X = 1

Cache

INTERCONNECTION (Bus, Multistage)

X = 1

Main Memory
2. MULTIPROCESSORS ARCHITECTURE

Distributed Write Cache Coherence

CPU1

Cache

CPU2

Cache

INTERCONNECTION (Bus, Multistage)

Main Memory
3. SIMPLE SPIN LOCKs

Two approaches

- Spin on Test and Set
- Spin on Read (Test and Test and Set)

GOOD

Easy to implement

BAD

Poor performance
3. SIMPLE SPIN LOCKs

Spin on Test and Set

lock := CLEAR
while( TestAndSet( lock ) = BUSY ) <= waiting
<Critical Section>
lock := CLEAR

Good performance with
- Few processors, Frequent polling, small critical section

Problems
- With many processors, lock holder must contend with others
- In bus type, consuming internetwork bandwidth
3. SIMPLE SPIN LOCKs

Spin on Test and Set

TestAndSet -> Critical section
3. SIMPLE SPIN LOCKs

Spin on Read (Test and Test and Set)

\[
\text{lock := CLEAR} \\
\text{while (lock = Busy or TestAndSet(lock) = Busy) <= waiting} \\
\text{<Critical Section>} \\
\text{lock := CLEAR}
\]

**Good**
- Reduce the cost of spin-waiting (If busy, don’t TestAndSet)
- With cache, better performance

**Bad**
- If the critical section is really small, same performance with Spin on TestAndSet
3. SIMPLE SPIN LOCKs

Spin on Read

lock=busy

TestAndSet

Critical section
3. SIMPLE SPIN LOCKs

Measurement of Performance

- SPIN on Test and Set
- SPIN on READ
- Ideal
3. SIMPLE SPIN LOCKs

Measurement of quiesce time

Fig. 2. Time to quiesce, spin on read (microseconds).
3. SIMPLE SPIN LOCKS

SPIN on Read problems with cache coherent

- LOCK Holder
- Cache
- CPU2
- CPU3
- Release Lock
- INTERCONNECTION (Bus, multistage)
- Main Memory
3. SIMPLE SPIN LOCKs

Source of problems: Contention

Contamination

CPU1

CPU2

Waiting

Request

NO Contention

CPU1

CPU2
4. SOFTWARE ALTERNATIVES

Four Software alternatives based on CSMA

Basic Idea
Inserting delays, reduce contention

Where?
• After the lock is released
• After every access to the memory

How long?
• Static (Slot)
• Dynamic (Backoff)

• Slots
• * in a few slots and few spinning processors, good
• * a few slots and many spinning processors, bad
WHERE? After the lock is released
Insert between release and try Test and Set

While (lock = busy or TestAndSet(lock) = busy)
{
    While(lock = busy);
    Delay();
}

While (lock = busy or Delay() or lock = busy or TestAndSet(lock) = busy)

• With many processors, performance good
• Only one processors, gives unnecessary delay.
While (lock=busy or \texttt{Delay()} or lock = busy or \texttt{TestAndSet(lock) = busy})
Where? After every access to the memory

```java
While (lock = busy or TestAndSet(lock) = busy)
{
    Delay();
}
```

- It checks less frequently
While (lock=busy or TestAndSet(lock) = busy) Delay();
4. SOFTWARE ALTERNATIVES

How long? Statically assigned slots to delay

<table>
<thead>
<tr>
<th>Processors</th>
<th>CPU</th>
<th>CPU</th>
<th>CPU</th>
<th>CPU</th>
<th>CPU</th>
<th>CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Delay Slots</td>
<td>0</td>
<td>3</td>
<td>5</td>
<td>7</td>
<td>9</td>
<td>11</td>
</tr>
</tbody>
</table>

Interconnection

Many spinning processors, good performance
Few spinning processors, bad performance
4. SOFTWARE ALTERNATIVES

Varied number of slots

GOOD

CPU  CPU  CPU  CPU  CPU

0  3  5

Interconnection

BAD

CPU  CPU  CPU  CPU  CPU

0  3  5

Interconnection
4. SOFTWARE ALTERNATIVES

Measurement overhead of Varied number of slots
How long? Dynamically CSMA backoff

While (TestAndSet(lock) = busy)
{
    value += random Backoff()
    if(value > max) value = max;
    Delay(value);
}
4. SOFTWARE ALTERNATIVES

Summary of New software alternatives

- Still remains contention
- Architecture with cache
  Good: less consuming inter-communication bandwidth
  Bad: quiescence by invalidation
- Inappropriately long delay gives bad performance
5. QUEUEING APPROACH

Basic Idea

- Each arriving processor enqueues itself
- Spin on a separate flag (in different location)
- When the processor releases the lock, dequeues itself.
- Set the flag of the next processor in the queue

GOOD

- Reduce invalidation, notify only for next processor

BAD

- Maintaining the queue is expensive, needed to be locked itself
- For small critical section, bad performance
5. QUEUEING APPROACH

Advanced Idea

- It needs a single atomic operation support (If not, it should be locked itself)
- Each arriving processor have a unique sequence number using atomic read-and-increment operation
- When the lock holder releases the lock, it taps the processor with next highest sequence number
Main code for queueing

Init
flags[0] := HAS_LOCK;
flags[1..P-1] := MUST_WAIT;
queueLast := 0;

Seq
myPlace := ReadAndIncrement(queueLast);

Wait
while (flags[myPlace mod P] = MUST_WAIT);

Lock

<critical section>

Unlock
flags[myPlace mod P] := MUST_WAIT;
flags[(myPlace+1) mod P] := HAS_LOCK
5. QUEUEING APPROACH

Queueing Mechanism

- Total processors = 8
- Critical Section = 1
- Spinning = 5
- queueLast = 6
- Next myPlace will be 7
5. QUEUEING APPROACH

Software approach

Queueing

lock=busy

Delay()

lock=busy

TestAndSet

myPlace=busy

myPlace=busy

Delay()

lock=busy

TestAndSet
5. QUEUEING APPROACH

Performance of Queueing approach

It has different performance according to each architectures

For multistage networks architecture, if each flag is on different memory module, it has good performance

For architecture without cache, all spinning processor access memory for checking their flags. It generation communication contention.

If there is no contention, poor performance relative to other approaches because of maintaining queue
6. MEASUREMENT RESULTS

Spinlock alternatives

![Graph showing overhead vs number of processors for different spinlock alternatives]
6. MEASUREMENT RESULTS

Normalized Overhead

![Graph showing normalized overhead with respect to the number of processors. The graph includes lines for 'spin test&set', 'spin on read', 'static ref.', 'backoff ref.', and 'queue'.]
7. HARDWARE SOLUTION

It still has tradeoffs

If the mechanism is specialized for spin lock performance, How about normal memory references?

Their suggestions
- Multistage Interconnection
- Bus interconnection
7. HARDWARE SOLUTION

Multistage Interconnection

Combining Networks
- Combine requests to same lock
- This has benefit with lots of contention

Hardware Queuing
- Blocking enter and exit instructions queue processes at memory module
- Eliminate polling across the network

Goodman’s Queue Links
- Hardware support to tap the next processor,
7. HARDWARE SOLUTION

Bus Interconnection

For TestAndSet, Use additional bus with write broadcast coherence. It keep current value, and Not make bus contention

Invalidate cache copies only when Test-and-Set succeeds

Read broadcast
• Eliminates the cascade of read-misses

Special handling of Test-and-Set
• if the lock is busy, manipulate like SPIN on read.
• Make “Test and Set” Fail as a read instruction
Spin-lock has different performance according to hardware architectures.

In software alternatives, Dynamic Ethernet back-off shows good performance.

In queueing approach, it shows poor performance with fewer spinning processors. But, it has good performance in lots of contention.

Performance can be increased with hardware support.