Lynx: Using OS and Hardware Support for Fast Fine-Grained Inter-Core Communication

Konstantina Mitropoulou, Vasileios Porpodas, Xiaochun Zhang and Timothy M. Jones

Computer Laboratory

ICS 2016, Istanbul
Outline

- Background:
  - Lamport’s queue
  - Multi-section queue
- Lynx queue
- Performance evaluation
Lamport’s Queue Bottlenecks

- Frequent thread synchronisation
- Cache ping-pong
Lamport’s Queue Bottlenecks

while (next_enqueue_ptr == dequeue_ptr) {

enqueue_ptr

depqueue_ptr

while (next_enqueue_ptr == dequeue_ptr) {

; ;

}
Lamport’s Queue Bottlenecks

\[
\text{enqueue\_ptr} \\
\text{dequeue\_ptr}
\]

\[
\text{while (next\_enqueue\_ptr == dequeue\_ptr)}
\]

Performance degradation due to:

- Frequent thread synchronisation
- Cache ping-pong
Lamport’s Queue Bottlenecks

```c
while (next_enqueue_ptr == dequeue_ptr) {
}
```

Performance degradation due to:
- Frequent thread synchronisation
Lamport’s Queue Bottlenecks

Performance degradation due to:

- Frequent thread synchronisation
- Cache ping-pong

```
while (next_enqueue_ptr == dequeue_ptr) {; }
```
Cache Ping-Pong

\[
\text{while}(\text{next\_enqueue\_ptr} == \text{dequeue\_ptr})\{;\}
\]
Cache Ping-Pong

while (next_enqueue_ptr == dequeue_ptr) {
    
    • Queue pointers ping-pong across cache hierarchy
Cache Ping-Pong

while(next_dequeue_ptr == enqueue_ptr) {
    
- Queue pointers ping-pong across cache hierarchy


Multi-Section Queue (MSQ): state-of-the-art

| section 1 | section 2 |

- Each section is exclusively used by one thread
Multi-Section Queue (MSQ): state-of-the-art

- Each section is exclusively used by one thread
Multi-Section Queue (MSQ): state-of-the-art

- Enqueue thread cannot access section 1 because dequeue thread still uses it
Multi-Section Queue (MSQ): state-of-the-art

- Enqueue thread cannot access section 1 because dequeue thread still uses it.
- Enqueue thread waits (spins) at the end of section 2.
Multi-Section Queue (MSQ): state-of-the-art

- Dequeue thread reached the end of section 1
Multi-Section Queue (MSQ): state-of-the-art

- Dequeue thread reached the end of section 1
- Enqueue thread enters section 1

enqueue_ptr

section 1

section 2
dequeue_ptr
Multi-Section Queue (MSQ): state-of-the-art

Performance optimisations:

- Infrequent boundary checks (less frequent synchronisation)
- Reduced cache ping-pong
Multi-Section Queue (MSQ): state-of-the-art

Performance optimisations:

- Infrequent boundary checks (less frequent synchronisation)
Multi-Section Queue (MSQ): state-of-the-art

Performance optimisations:

- Infrequent boundary checks (less frequent synchronisation)
- Reduced cache ping-pong
MSQ Control-Flow Graph and Internals

enqueue function

```
1
2
3
4
5
6
```

dequeue function

```
1
2
3
4
5
```
MSQ Control-Flow Graph and Internals

enqueue function

enqueue

Slide 11 of 32

http://www.cl.cam.ac.uk/~km647/
MSQ Control-Flow Graph and Internals

enqueue function

synchronisation code

enqueue

1
2
3
4
5
6
enqueue function

enqueue

checks if next section is free

synchronisation code

enqueue
enqueue function

enqueue

checks if next section is free

spin loop

synchronisation code
enqueue function

1
2
3
4
5
6

enqueue

checks if next section is free

spin loop

update local variables

synchronisation code
MSQ Control-Flow Graph and Internals

enqueue function

enqueue

spin loop

update local variables

update shared variable

synchronisation code

checks if next section is free
MSQ Control-Flow Graph and Internals

enqueue

checks if next section is free

spin loop

update local variables

update shared variable

join basic-block

esynchronisation code

enqueue function
MSQ Control-Flow Graph and Internals

section 1

enqueue_ptr

section 2

dehqueue_ptr

synchronisation code
enqueue function

lea rax, [rdx+8]
mov QWORD PTR [rdx], rcx
mov rdx, rax
and rdx, ROTATE MASK
test eax, SECTION_MASK
jne .L2

enqueue
enqueue function

enqueue

synchronisation code

lea rax, [rdx+8]
mov QWORD PTR [rdx], rcx
mov rdx, rax
and rdx, ROTATE MASK
test eax, SECTION_MASK
jne .L2
incr pointer

slide 13 of 32
enqueue function

1. enqueue

2. lea rax, [rdx+8]
3. mov QWORD PTR [rdx], rcx
4. mov rdx, rax
5. and rdx, ROTATE MASK
6. test eax, SECTION_MASK
7. jne .L2

 incr pointer
 store

synchronisation code

slide 13 of 32
enqueue function

synchronisation code

enqueue

lea rax, [rdx+8]
mov QWORD PTR [rdx], rcx
mov rdx, rax
and rdx, ROTATE MASK
test eax, SECTION_MASK
jne .L2

incr pointer
store
compiler’s copy

slide 13 of 32
http://www.cl.cam.ac.uk/~km647/
enqueue function

1. lea rax, [rdx+8]
2. mov QWORD PTR [rdx], rcx
3. mov rdx, rax
4. and rdx, ROTATE MASK
5. test eax, SECTION_MASK
6. jne .L2

enqueue

synchronisation code

rotate pointer

incr pointer

store

compiler’s copy

rotate pointer

store

compiler’s copy

rotate pointer

store

compiler’s copy

rotate pointer

store

compiler’s copy
MSQ Control-Flow Graph and Internals

enqueue function

1. lea rax, [rdx+8]
2. mov QWORD PTR [rdx], rcx
3. mov rdx, rax
4. and rdx, ROTATE MASK
5. test eax, SECTION_MASK
6. jne .L2

enqueue

synchronisation code

rotate pointer

compiler’s copy

store

end of section

incr pointer

slide 13 of 32

http://www.cl.cam.ac.uk/~km647/
enqueue function

1. lea rax, [rdx+8]
2. mov QWORD PTR [rdx], rcx
3. mov rdx, rax
4. and rdx, ROTATE MASK
5. test eax, SECTION_MASK
6. jne .L2

enqueue

synchronisation code

end of section

compiler’s copy

rotate pointer

store

incr pointer

skip sync code

end of section
Optimal Queue

Optimal queue features:

- infinite size
Optimal Queue

Optimal queue features:

• infinite size
• 2 instructions overhead
  1. pointer increment
  2. store into the queue
Lynx: Just 2 instructions overhead

Lynx removes part of enqueue (boundary checks) and all the synchronisation overhead off the critical path.
Lynx(1): H/W triggered Synchronisation

enqueue function

1. lea rax, [rdx+8]
2. mov QWORD PTR [rdx], rcx
3. mov rdx, rax
4. and rdx, ROTATE MASK
5. test eax, SECTION_MASK
6. jne .L2
Lynx(1): H/W triggered Synchronisation

enqueue function

1. lea rax, [rdx+8]
2. mov QWORD PTR [rdx], rcx
3. mov rdx, rax
4. and rdx, ROTATE MASK
5. test eax, SECTION_MASK
6. jne .L2
Lynx(1): H/W triggered Synchronisation

enqueue function

lea rax, [rdx+8]
mov QWORD PTR [rdx], rcx
mov rdx, rax
and rdx, ROTATE MASK

enqueue

test eax, SECTION_MASK
jne .L2
Lynx(1): H/W triggered Synchronisation

| section 1 | section 2 |

- A red zone is a non-read and non-write part of memory
- SSRZ: Section Synchronisation Red-Zone
Lynx(1): H/W triggered Synchronisation

- A red zone is a non-read and non-write part of memory
Lynx(1): H/W triggered Synchronisation

- SSRZ: Section Synchronisation Red-Zone

A red zone is a non-read and non-write part of memory.
Lynx(1): H/W triggered Synchronisation

- Whether the SIGSEGV is from the queue or the system
- Which thread raised the exception
- If the thread is in section 1 or 2
- If the next section is free
Lynx(1): H/W triggered Synchronisation

![Diagram showing sections and pointers](http://www.cl.cam.ac.uk/~km647/)

- whether the SIGSEGV is from the queue or the system
- which thread raised the exception
- if the thread is in section 1 or 2
- if the next section is free
Lynx(1): H/W triggered Synchronisation

- Whether the SIGSEGV is from the queue or the system
- Which thread raised the exception
- If the thread is in section 1 or 2
- If the next section is free
Lynx(1): H/W triggered Synchronisation

Lynx’s handler checks:

- whether the SIGSEGV is from the queue or the system
- which thread raised the exception
- if the thread is in section 1 or 2
- if the next section is free
Lynx(1): H/W triggered Synchronisation

Lynx’s handler checks:

- whether the SIG_SEGV is from the queue or the system
Lynx(1): H/W triggered Synchronisation

Lynx’s handler checks:

- whether the SIG_SEGV is from the queue or the system
- which thread raised the exception
Lynx(1): H/W triggered Synchronisation

Lynx’s handler checks:

- whether the SIG_SEGV is from the queue or the system
- which thread raised the exception
- if the thread is in section 1 or 2
Lynx(1): H/W triggered Synchronisation

Lynx’s handler checks:

- whether the SIG_SEGV is from the queue or the system
- which thread raised the exception
- if the thread is in section 1 or 2
- if the next section is free
Lynx(1): H/W triggered Synchronisation

![Diagram of Lynx(1) hardware triggered synchronization]

- `enqueue_ptr`
- `dequeue_ptr`
- `SSRZ`
Lynx(1): H/W triggered Synchronisation

section 1

SSRZ

devqueue_ptr

enqueue_ptr

section 2

SSRZ

dequeue_ptr
Lynx(2): H/W triggered Pointer Rotation

enqueue function

synchronisation code

enqueue

lea rax, [rdx+8]
mov QWORD PTR [rdx], rcx
mov rdx, rax
and rdx, ROTATE MASK
test eax, SECTION_MASK
jne .L2
Lynx(2): H/W triggered Pointer Rotation

enqueue function

enqueue

synchronisation code

lea rax, [rdx+8]
mov QWORD PTR [rdx], rcx
mov rdx, rax
and rdx, ROTATE MASK
test eax, SECTION_MASK
jne .L2
Lynx(2): H/W triggered Pointer Rotation

- SSRZ: Section Synchronisation Red-Zone
Lynx(2): H/W triggered Pointer Rotation

- SSRZ: Section Synchronisation Red-Zone
- PRRZ: Pointer Rotation Red-Zone
Lynx(2): H/W triggered Pointer Rotation

- SSRZ: Section Synchronisation Red-Zone
- PRRZ: Pointer Rotation Red-Zone
Lynx(2): H/W triggered Pointer Rotation

- SSRZ: Section Synchronisation Red-Zone
- PRRZ: Pointer Rotation Red-Zone
Lynx(2): H/W triggered Pointer Rotation

- SSRZ: Section Synchronisation Red-Zone
- PRRZ: Pointer Rotation Red-Zone
Lynx(2): H/W triggered Pointer Rotation

Two types of red-zones:
Lynx(2): H/W triggered Pointer Rotation

Two types of red-zones:

1. moving red-zone: SSRZ (Section Synchronisation Red-Zone)

SSRZ SSRZ PRRZ

dequeue_ptr
enqueue_ptr

section 1 section 2
Lynx(2): H/W triggered Pointer Rotation

Two types of red-zones:

1. moving red-zone: SSRZ (Section Synchronisation Red-Zone)
2. fixed red-zone: PRRZ (Pointer Rotation Red-Zone)
Experimental Setup

- Implementation in C++ with inline assembly
Experimental Setup

- Implementation in C++ with inline assembly
- Evaluation on several machines: from embedded SOCs to server CPUs
Experimental Setup

- Implementation in C++ with inline assembly
- Evaluation on several machines: from embedded SOCs to server CPUs
- Throughput experiments for a wide range of queue sizes
Experimental Setup

- Implementation in C++ with inline assembly
- Evaluation on several machines: from embedded SOCs to server CPUs
- Throughput experiments for a wide range of queue sizes
- Absolute throughput performance in GB/s
Throughput (GB/s) on Intel core-i5

Throughput for 64bit Memory Instr. (Core-i5 4570)

Queue size

GB/s

Queue size
Throughput (GB/s) on Various Machines

Throughput for 64bit Memory Instr. (Xeon E5-2667v2)

Throughput for 64bit Memory Instr. (Opteron 6376)

Throughput for 64bit Memory Instr. (Core-i3 2367M)

Throughput for 64bit Memory Instr. (Celeron J1900)
The best queue configuration with Lynx is better than the best with MSQ
Conclusion

- Proposed Lynx: a lock-free SP/SC software queue with just 2 instructions overhead
Conclusion

- Proposed Lynx: a lock-free SP/SC software queue with just 2 instructions overhead
- Relies on existing commodity H/W and O/S support for memory protection
Conclusion

- Proposed Lynx: a lock-free SP/SC software queue with just 2 instructions overhead
- Relies on existing commodity H/W and O/S support for memory protection
- The overhead of synchronisation and boundary checking is moved to the exception handler
Conclusion

- Proposed Lynx: a lock-free SP/SC software queue with just 2 instructions overhead
- Relies on existing commodity H/W and O/S support for memory protection
- The overhead of synchronisation and boundary checking is moved to the exception handler
- Throughput increases by up to 57%

http://www.cl.cam.ac.uk/~km647/
Conclusion

- Proposed Lynx: a lock-free SP/SC software queue with just 2 instructions overhead
- Relies on existing commodity H/W and O/S support for memory protection
- The overhead of synchronisation and boundary checking is moved to the exception handler
- Throughput increases by up to 57%

https://www.cl.cam.ac.uk/~km647/papers/lynx/lynxQ.tar.bz2
Back-up slides
Throughput (GB/s) on Intel core-i5

[Graphs showing throughput for different memory sizes and operations like MSQ-mov and Lynx-mov]
Breakdown of Synchronisation Overheads

![Diagram showing the breakdown of synchronisation overheads](http://www.cl.cam.ac.uk/~km647/)

- **enqueue** and **dequeue**
- **Inter-Core Latency**
- **Kernel Overhead**
- **Synchronisation (in spin-lock)**
- **Real Execution**

---

Slide 31 of 32
Throughput (GB/s) on Different Machines