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

UKMAC 2016, Edinburgh
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

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

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

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

Performance degradation due to:

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

\[ \text{while(} \text{next}\_\text{enqueue}\_\text{ptr} == \text{dequeue}\_\text{ptr}\text{)} \{ \; ; \} \]

Performance degradation due to:

- Frequent thread synchronisation
Lamport’s Queue Bottlenecks

while (next_enqueue_ptr == dequeue_ptr) {
}

Performance degradation due to:

- Frequent thread synchronisation
- Cache ping-pong
Cache Ping-Pong

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

while (next_enqueue_ptr == dequeue_ptr) {};

- Queue pointers ping-pong across cache hierarchy
Cache Ping-Pong

```c
while (next_dequeue_ptr == enqueue_ptr) {
};
```

- Queue pointers ping-pong across cache hierarchy
Multi-Section Queue (MSQ): state-of-the-art
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
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

depqueue function

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

enqueue function

enqueue
enqueue function

enqueue

checks if next section is free

synchronisation code
MSQ Control-Flow Graph and Internals

enqueue function

enqueue

spin loop

checks if next section is free

synchronisation code

enqueue function
MSQ Control-Flow Graph and Internals

1. enqueue function
2. enqueue
3. spin loop
4. update local variables
5. checks if next section is free
6. synchronisation code

enqueue function
MSQ Control-Flow Graph and Internals

enqueue function

1. enqueue
2. checks if next section is free
3. spin loop
4. update local variables
5. update shared variable
6. enqueue function

synchronisation code
MSQ Control-Flow Graph and Internals

enqueue function

1. enqueue
2. checks if next section is free
3. spin loop
4. update local variables
5. update shared variable
6. join basic-block

synchronisation code
MSQ Control-Flow Graph and Internals

depqueue_ptr

enqueue_ptr

section 1

section 2

synchronisation code

slide 12 of 30
http://www.cl.cam.ac.uk/~km647/
enqueue function

1
2
3
4
5
6

enqueue

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

synchronisation code

slide 13 of 30
http://www.cl.cam.ac.uk/~km647/
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 30

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

synchronisation code

store
incr pointer
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

incr pointer

store

compiler’s copy

slide 13 of 30
enqueue function

enqueue

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

rotate pointer
compiler's copy
store
incr pointer

slide 13 of 30
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

synchronisation code

end of section

rotate pointer

store

compiler’s copy

rotate pointer

end of section
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

synchronisation code

enqueue

incr pointer
store
compiler’s copy
rotate pointer
end of section
skip sync code
Optimal Queue

Optimal queue features:

- infinite size
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

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

<table>
<thead>
<tr>
<th>section 1</th>
<th>section 2</th>
</tr>
</thead>
</table>

- 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
Lynx(1): H/W triggered Synchronisation

The diagram illustrates the synchronization between two sections, section 1 and section 2. Theenqueue_ptr and dequeue_ptr are shown as arrows indicating the direction of synchronization. SSRZ (Synchronisation and Synchronization) is indicated in both sections.

- 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

The diagram represents a hardware-triggered synchronization mechanism in Lynx(1).
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

Slide 18 of 30

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

- The dequeue thread still uses the first section.
- The enqueue thread waits at the end of the second section and it adds a new red zone.
- The new red zone is part of the synchronisation and it is temporarily added.
Lynx(1): H/W triggered Synchronisation

- The dequeue thread has finished with the first section
- The enqueue thread removes the second red zone and it enters the first section
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

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

enqueue_ptr

section 1
SSRZ

dequeue_ptr

section 2
SSRZ

enqueue_ptr
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:

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

Two types of red-zones:

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

SSRZ  SSRZ  PRRZ
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 a wide range of machines: from embedded SOCs to server CPUs
Experimental Setup

- Implementation in C++ with inline assembly
- Evaluation on a wide range of 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 a wide range of 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

Throughput for 64bit Memory Instr. (Core-i5 4570)
Breakdown of Lynx Overheads

![Bar chart showing the breakdown of Lynx overheads for different queue sizes. The chart includes categories for real, kernel, sync, handler, and other execution times. Each bar is divided into segments representing the percentage of execution time for each category.]
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%
Source Code

https://www.cl.cam.ac.uk/~km647/papers/lynx/lynxQ.tar.bz2

or

https://www.repository.cam.ac.uk/handle/1810/254651