“A Wait-free Queue as Fast as Fetch-and-Add”

Chaoran Yang, John Mellor-Crummey

Loris Lucido

17 mars 2017

Progress guarantee

Blocking queue → unexpected delays from the Operating System can block every threads.

Progress Guarantee

	An operation is guaranteed to end in a finite number of steps for...
• *Obstruction-free*	*1 thread only (in isolation)*
• *Lock-free*	*at least 2 threads*
• *Wait-free*	*any number of threads*

Implementation - Atomic primitives

Fetch-and-Add

$FAA(x, v)$ returns the value of x and increments it by v

Compare-and-Swap

$CAS(x, t, v)$ replaces x by v if x equals t

Problematic

Can an **obstruction-free/lock-free** queue using **fetch-and-add** can be transformed to a **wait-free** queue using the **fast-path-slow-path** methodology ?

Obstruction-free queue

An obstruction-free queue using an infinite array.

```
⊤: invalid cell
⊥: empty cell
enqueue(x: var) {
  do t := FAA(&Tail, 1);
  while (!CAS(&Queue[t], ⊥,
       x));
}
```

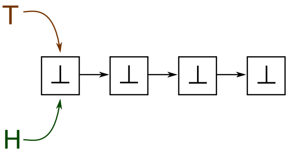
```
dequeue(x: var) {
  do h := FAA(&Head, 1);
  while (CAS(&Queue[h], ⊥, ⊤) and
      Tail > h);
  return (Queue[h] == ⊤ ? EMPTY :
      Queue[h]);
}
```

Obstruction-free queue

An obstruction-free queue using an infinite array.

```
⊤: invalid cell
⊥: empty cell
enqueue(x: var) {
  do t := FAA(&Tail, 1);
  while (!CAS(&Queue[t], ⊥,
        x));
}
```

```
dequeue(x: var) {
  do h := FAA(&Head, 1);
  while (CAS(&Queue[h], ⊥, ⊤) and
      Tail > h);
  return (Queue[h] == ⊤ ? EMPTY :
      Queue[h]);
}
```

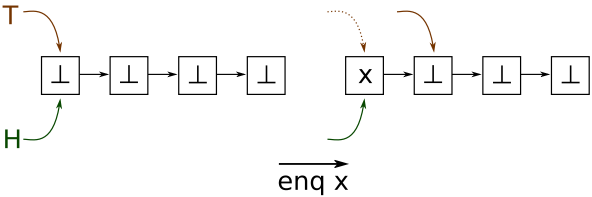


Obstruction-free queue

An obstruction-free queue using an infinite array.

```
⊤: invalid cell
⊥: empty cell
enqueue(x: var) {
  do t := FAA(&Tail, 1);
  while (!CAS(&Queue[t], ⊥,
        x));
}
```

```
dequeue(x: var) {
  do h := FAA(&Head, 1);
  while (CAS(&Queue[h], ⊥, ⊤) and
       Tail > h);
  return (Queue[h] == ⊤ ? EMPTY :
       Queue[h]);
}
```

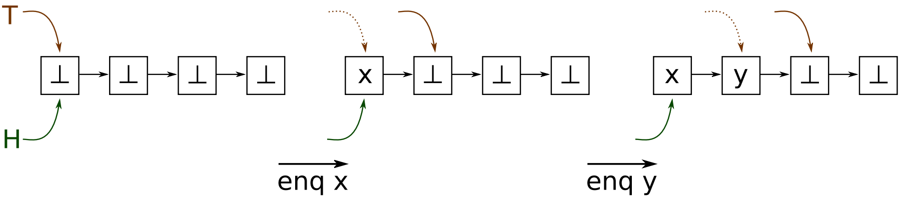


Obstruction-free queue

An obstruction-free queue using an infinite array.

```
⊤: invalid cell
⊥: empty cell
enqueue(x: var) {
  do t := FAA(&Tail, 1);
  while (!CAS(&Queue[t], ⊥,
        x));
}
```

```
dequeue(x: var) {
  do h := FAA(&Head, 1);
  while (CAS(&Queue[h], ⊥, ⊤) and
       Tail > h);
  return (Queue[h] == ⊤ ? EMPTY :
       Queue[h]);
}
```

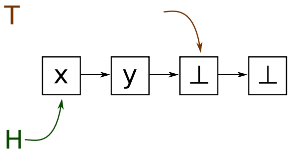


Obstruction-free queue

An obstruction-free queue using an infinite array.

```
⊤: invalid cell
⊥: empty cell
enqueue(x: var) {
  do t := FAA(&Tail, 1);
  while (!CAS(&Queue[t], ⊥,
        x));
}
```

```
dequeue(x: var) {
  do h := FAA(&Head, 1);
  while (CAS(&Queue[h], ⊥, ⊤) and
        Tail > h);
  return (Queue[h] == ⊤ ? EMPTY :
        Queue[h]);
}
```

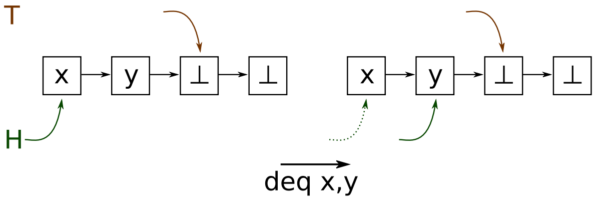


Obstruction-free queue

An obstruction-free queue using an infinite array.

```
⊤: invalid cell
⊥: empty cell
enqueue(x: var) {
  do t := FAA(&Tail, 1);
  while (!CAS(&Queue[t], ⊥,
       x));
}
```

```
dequeue(x: var) {
  do h := FAA(&Head, 1);
  while (CAS(&Queue[h], ⊥, ⊤) and
      Tail > h);
  return (Queue[h] == ⊤ ? EMPTY :
      Queue[h]);
}
```

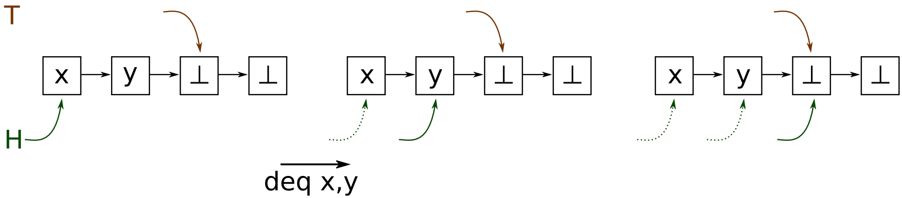


Obstruction-free queue

An obstruction-free queue using an infinite array.

```
⊤: invalid cell
⊥: empty cell
enqueue(x: var) {
  do t := FAA(&Tail, 1);
  while (!CAS(&Queue[t], ⊥,
       x));
}
```

```
dequeue(x: var) {
  do h := FAA(&Head, 1);
  while (CAS(&Queue[h], ⊥, ⊤) and
       Tail > h);
  return (Queue[h] == ⊤ ? EMPTY :
       Queue[h]);
}
```

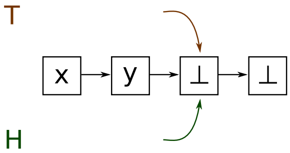


Obstruction-free queue

An obstruction-free queue using an infinite array.

```
⊤: invalid cell
⊥: empty cell
enqueue(x: var) {
  do t := FAA(&Tail, 1);
  while (!CAS(&Queue[t], ⊥,
        x));
}
```

```
dequeue(x: var) {
  do h := FAA(&Head, 1);
  while (CAS(&Queue[h], ⊥, ⊤) and
       Tail > h);
  return (Queue[h] == ⊤ ? EMPTY :
       Queue[h]);
}
```

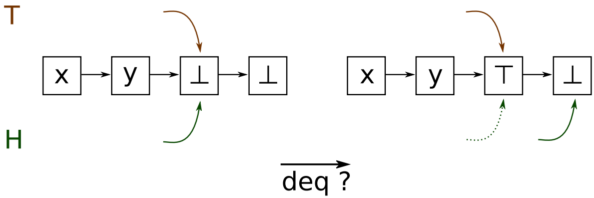


Obstruction-free queue

An obstruction-free queue using an infinite array.

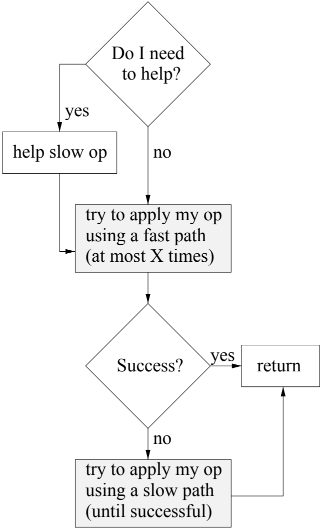
```
⊤: invalid cell
⊥: empty cell
enqueue(x: var) {
  do t := FAA(&Tail, 1);
  while (!CAS(&Queue[t], ⊥,
        x));
}
```

```
dequeue(x: var) {
  do h := FAA(&Head, 1);
  while (CAS(&Queue[h], ⊥, ⊤) and
       Tail > h);
  return (Queue[h] == ⊤ ? EMPTY :
       Queue[h]);
}
```

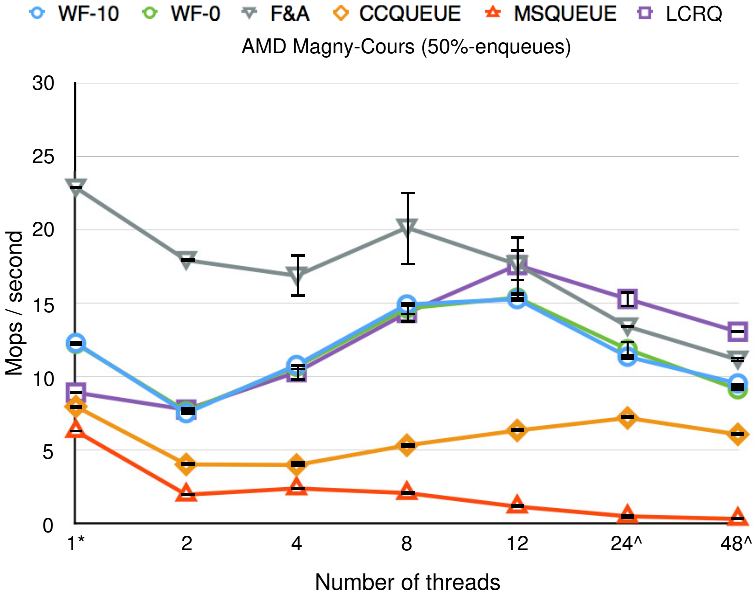


Fast-path-slow-path methodology

Transform lock-free/obstruction-free queue into wait-free queue


Do I need to help?
yes
no
help slow op
try to apply my op using a fast path (at most X times)
Success?
yes
return
no
try to apply my op using a slow path (until successful)

Results



WF-10
WF-0
F&A
CCQUEUE
MSQUEUE
LCRQ
AMD Magny-Cours (50%-enqueues)
Mops / second
30
25
20
15
10
5
0
1*
2
4
8
12
24^
48^
Number of threads

Discussions

Some lock-free/blocking objects are *practically* wait-free.

- 99.97% enqueue done in one try on fast-path
- 95.95% dequeue done in one try on fast-path

Low probability of the worst-case scenario (slow-path)

Conclusion

- First wait-free queue with performance as good as Lock-free queue

- Design complexity increased for wait-free queue

Thanks for your attention !

References I

- Dan ALISTARH, Keren CENSOR-HILLEL et Nir SHAVIT. “Are Lock-Free Concurrent Algorithms Practically Wait-Free ?” In : *J. ACM* 63.4 (sept. 2016), 31 :1-31 :20. ISSN : 0004-5411. DOI : 10.1145/2903136.

- Tudor DAVID et Rachid GUERRAOUI. “Concurrent Search Data Structures Can Be Blocking and Practically Wait-Free”. In : *Proceedings of the 28th ACM Symposium on Parallelism in Algorithms and Architectures*. SPAA ’16. New York, NY, USA : ACM, 2016, p. 337-348. ISBN : 978-1-4503-4210-0. DOI : 10.1145/2935764.2935774.

- Chaoran YANG et John MELLOR-CRUMMEY. “A Wait-free Queue As Fast As Fetch-and-add”. In : *SIGPLAN Not.* 51.8 (fév. 2016), 16 :1-16 :13. ISSN : 0362-1340. DOI : 10.1145/3016078.2851168.

- Adam MORRISON et Yehuda AFEK. “Fast Concurrent Queues for x86 Processors”. In : *SIGPLAN Not.* 48.8 (fév. 2013), p. 103-112. ISSN : 0362-1340. DOI : 10.1145/2517327.2442527.

- Panagiota FATOUROU et Nikolaos D. KALLIMANIS. “Revisiting the Combining Synchronization Technique”. In : *SIGPLAN Not.* 47.8 (fév. 2012), p. 257-266. ISSN : 0362-1340. DOI : 10.1145/2370036.2145849.

References II

Alex KOGAN et Erez PETRANK. “A Methodology for Creating Fast Wait-free Data Structures”. In : *SIGPLAN Not.* 47.8 (fév. 2012), p. 141-150. ISSN : 0362-1340. DOI : 10.1145/2370036.2145835.

Maged M. MICHAEL et Michael L. SCOTT. “Simple, Fast, and Practical Non-Blocking and Blocking Concurrent Queue Algorithms”. In : *Proc. 15th ACM Symp. on Principles of Distributed Computing*. 1996, p. 267-275.