LRSCwait: Enabling Scalable and Efficient Synchronization in Manycore Systems through Polling-Free and Retry-Free Operation

Samuel Riedel, Marc Gantenbein, Alessandro Ottaviano, Torsten Hoefler, and Luca Benini
Integrated Systems Laboratory (IIS) and Scalable Parallel Computing Lab (SPCL), ETH Zürich, Switzerland

Polling-free synchronization?

Concurrent algorithms require synchronization

- Blocking synchronization: Locks
  → Polling shared resources
- Non-blocking synchronization: CAS, LR/SC
  → Retry failed attempts

Polling limits performance

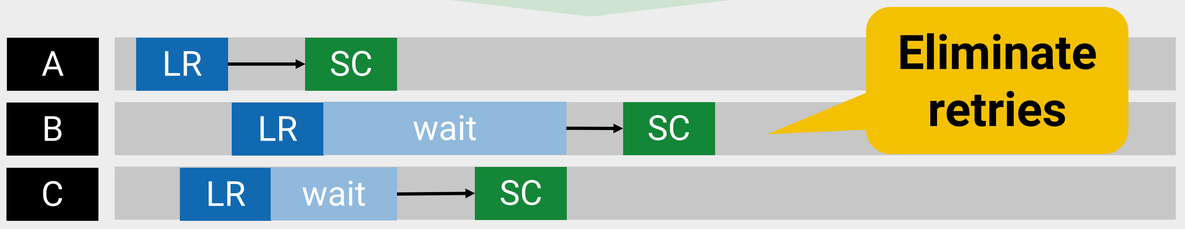
- Wasted work during unsuccessful attempts
- Contention for shared resources with cores doing work

Eliminate polling during synchronization

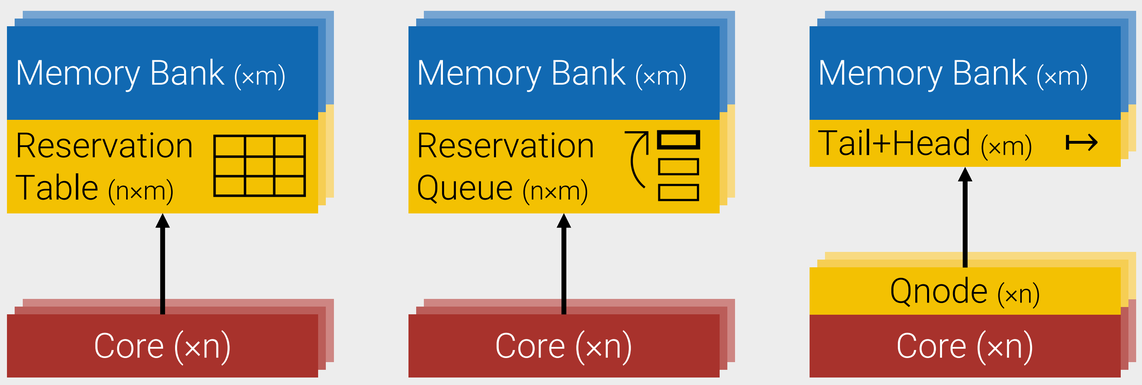
Move the linearization point from the SC to the LR

- Decide who 'wins' the SC already at the LR
- Only one core executes an LR/SC pair at once

LRSCwait



Efficient hardware implementation



LRSC

- Reservation Table for each memory
- Large overhead

LRSCwait Ideal

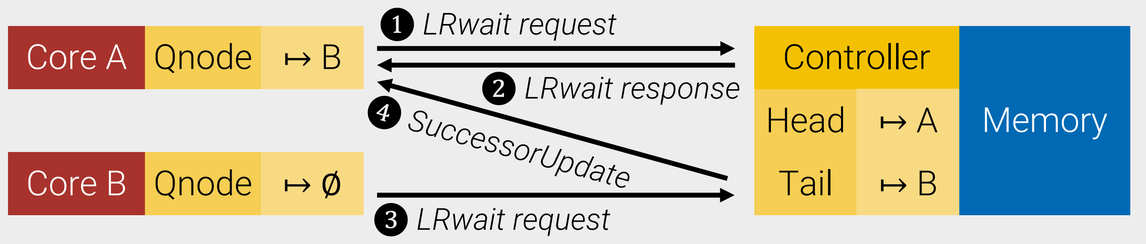
- Reservation Queue for each memory
- Large overhead

LRSCwait Colibri

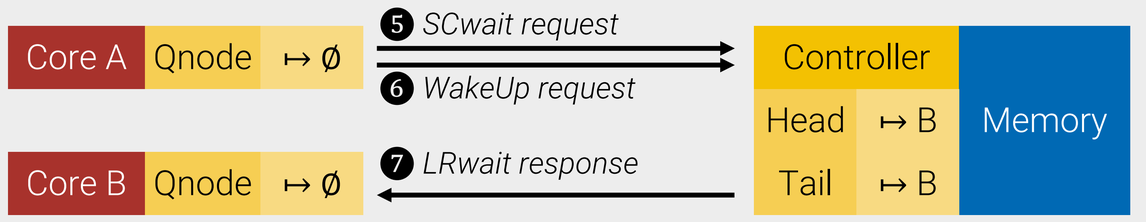
- Distributed Queue
  - Pointer at memory
  - Qnode at cores
- Small overhead

Colibri: Building a distributed queue

Core A and B issue a LRwait



Core A issues a SCwait

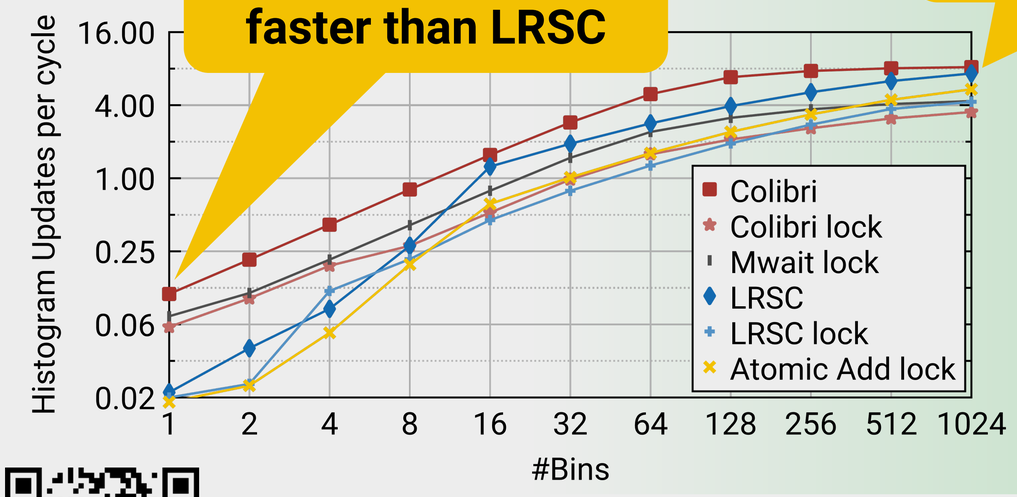


Implement and evaluate on MemPool

- A 256-core system with 1024 memory banks (1MiB)
- Implemented in GlobalFoundries' 22nm process
- Colibri comes with only a 6% area overhead

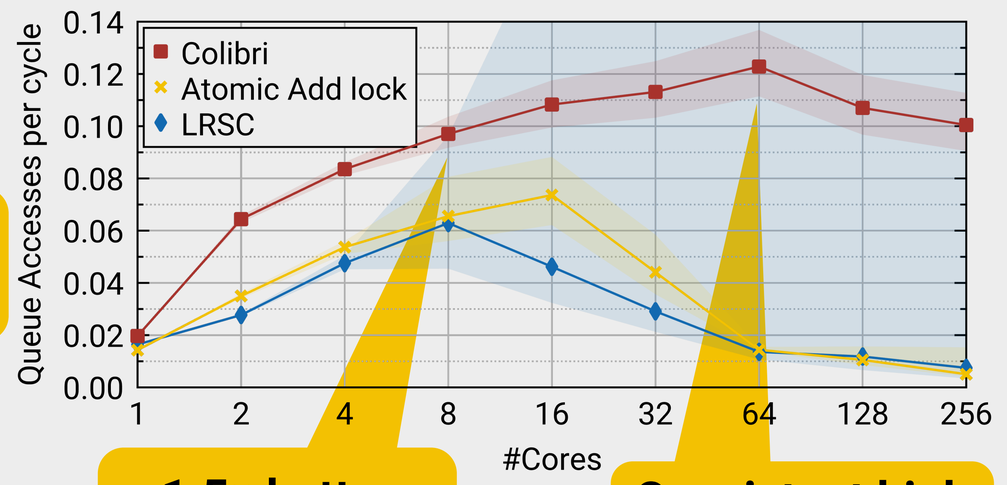
Colibri is up to 6.5× faster than LRSC

LRSCWait Colibri is 13% faster than LRSC



Evaluate the throughput of different atomic read-modify-write operations and lock implementations at varying levels of contention.

Compare concurrent queue implementations.



1.5× better performance

Consistent high throughput (9×)

Colibri improves energy efficiency by up to 7.1× compared to LRSC by eliminating polling and interference.

◁ open-source on GitHub!

sriedel@iis.ee.ethz.ch pulp-platform.org @pulp_platform