

Relativistic Programming

Josh Triplett, Paul E. McKenney, Phil Howard, Jonathan Walpole

Motivations

Memory/communication wall

- Speed of data limited to speed of light
 - $c/3\text{GHz} \approx 0.1$ meters/cycle (4 inches/cycle)
 - Ignores propagation delay, ramp time, speed of signals
- Hard physical limit on CPU–CPU communication

Throughput vs. Latency

- CPUs can use caches to work independently
- CPUs cannot reach agreement via communication in 1 cycle

To scale, we must reduce the need for agreement among CPUs

Principles

- By analogy with physics: no global reference frame
- Each thread works with “relative” view of memory in cache
- Tolerate lack of global operation order between threads
 - Per-thread ordering, partial-order constraints
- Tolerate conflicts: concurrent reads and writes

Comparison

Locking

- Fine-grained locking difficult
- High overhead
- Need to agree/coordinate

Transactional Memory

- Automatically fine-grained
- High overhead
- Need to agree/coordinate

Non-Blocking Synchronization

- Automatically fine-grained
- Hard to get right
- High overhead
- Need to agree/coordinate

Relativistic Programming

- No critical section
- Not excessively complex
- Minimal overhead
- No need to agree/coordinate

Examples

- Per-CPU variables: efficient update, approximate aggregation
- Deferred destruction — Read-Copy Update (RCU)

Read-Copy Update (RCU)

- Delimited readers with near-zero overhead
- “Wait for all current readers to finish” operation
- Primitives for conflict-tolerant operations
 - Assign pointer, dereference pointer
 - Avoid error-prone use of raw memory barriers
- Conventions:
 - Keep structures consistent at all times
 - Manipulate structure pointers; don’t mutate data
 - Reclaim memory only after potential readers finish
- Particularly good for read-mostly data structures

Properties

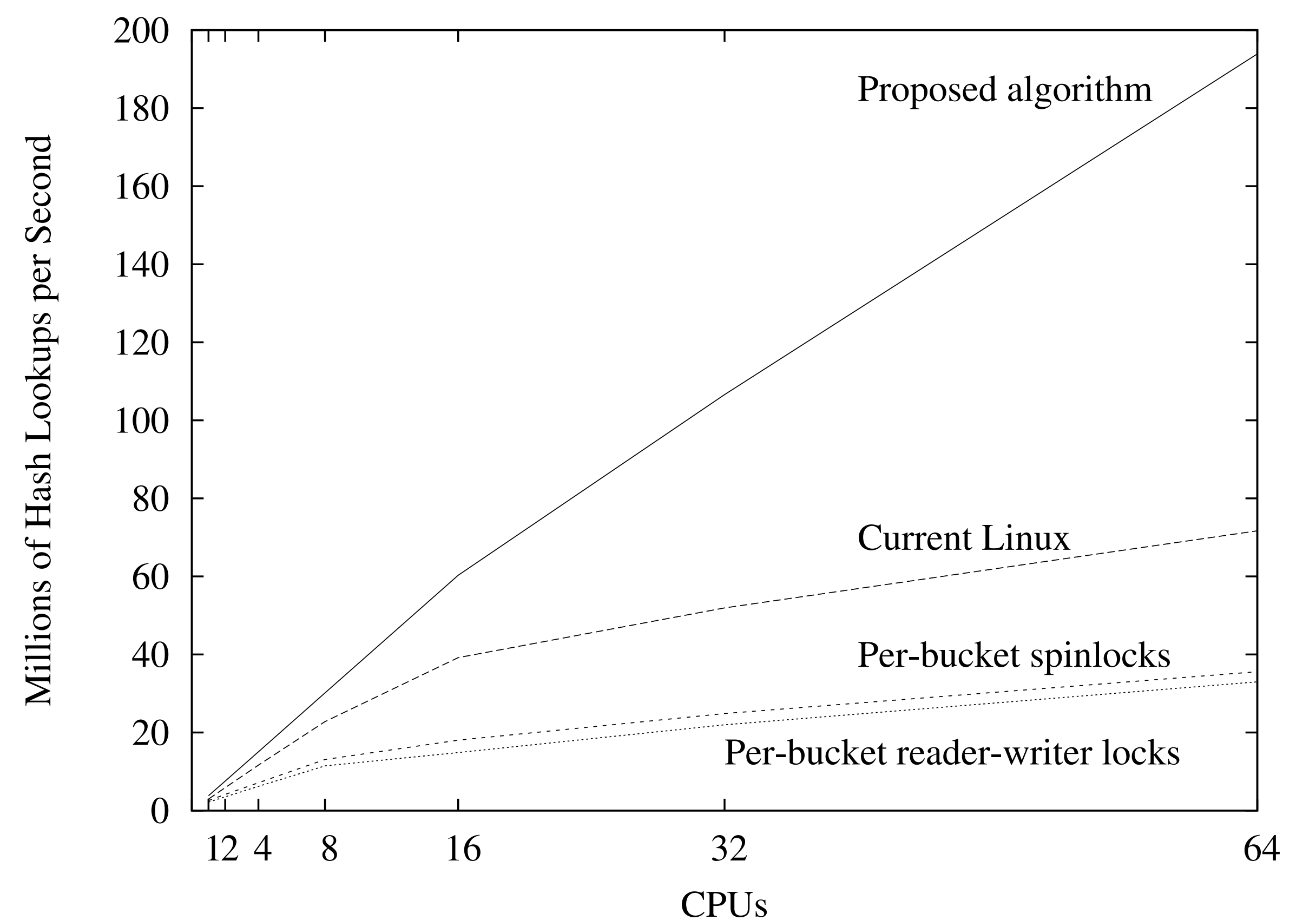
- Wait-free
- Linear scalability
- Minimal cache misses
- Near-zero synchronization overhead
- Usable anywhere: software/hardware interrupts, NMIs

RCU data structures

- Linked lists, hash tables, radix trees, Fibonacci tries
- Readers may see insertions and removals out of order
- Readers never see reclaimed memory

Recent hash table results

- New move operation
- 2-10x as scalable as current Linux
- Far more scalable than fine-grained locking
- Resizable hash tables developed; undergoing testing



RCU hash table lookups with concurrent moves, 999:1 ratio

In widespread use

- RCU implemented and used in Linux since 2002
- 2800+ uses in current Linux 2.6.31
- Userspace implementation liburcu now available

For more information

<http://wiki.cs.pdx.edu/rp/>
Josh Triplett <josh@joshtriplett.org>
Paul E. McKenney <paulmck@linux.vnet.ibm.com>
Phil Howard <pwh@cs.pdx.edu>
Jonathan Walpole <walpole@cs.pdx.edu>