Efficient Multi-Word Compare and Swap

Rachid Guerraoui
EPFL, Lausanne, Switzerland
rachid.guerraoui@epfl.ch

Alex Kogan
Oracle Labs, Burlington, MA, USA
alex.kogan@oracle.com

Virendra J. Marathe
Oracle Labs, Burlinton, MA, USA
virendra.marathe@oracle.com

Igor Zablotchi[1]
EPFL, Lausanne, Switzerland
igor.zablotchi@epfl.ch

Abstract

Atomic lock-free multi-word compare-and-swap (MCAS) is a powerful tool for designing concurrent algorithms. Yet, its widespread usage has been limited because lock-free implementations of MCAS make heavy use of expensive compare-and-swap (CAS) instructions. Existing MCAS implementations indeed use at least $2k + 1$ CASes per k-CAS. This leads to the natural desire to minimize the number of CASes required to implement MCAS.

We first prove in this paper that it is impossible to "pack" the information required to perform a k-word CAS (k-CAS) in less than k locations to be CASed. Then we present the first algorithm that requires $k + 1$ CASes per call to k-CAS in the common uncontended case. We implement our algorithm and show that it outperforms a state-of-the-art baseline in a variety of benchmarks in most considered workloads. We also present a durably linearizable (persistent memory friendly) version of our MCAS algorithm using only 2 persistence fences per call, while still only requiring $k + 1$ CASes per k-CAS.

2012 ACM Subject Classification Theory of computation → Concurrent algorithms

Keywords and phrases lock-free, multi-word compare-and-swap, persistent memory

Digital Object Identifier 10.4230/LIPIcs.DISC.2020.4

Related Version `https://arxiv.org/abs/2008.02527`

Funding This work has been supported in part by the European Research Council (ERC) Grant 339539 (AOC).

1 Introduction

Compare-and-swap (CAS) is a foundational primitive used pervasively in concurrent algorithms on shared memory systems. In particular, it is used extensively in *lock-free* algorithms, which avoid the pitfalls of blocking synchronization (e.g., that employs locks) and typically deliver more scalable performance on multicore systems. CAS conditionally updates a memory word such that a new value is written if and only if the old value in that word matches some expected value. CAS has been shown to be universal, and thus can implement any shared object in a non-blocking manner [32]. This primitive (or the similar load-linked/store-conditional (LL/SC)) is nowadays provided by nearly every modern architecture.

[1] This work was done when the author was an intern at Oracle Labs.