A Practical Multi-Word Compare-and-Swap Operation

Timothy L. Harris, Keir Fraser and Ian A. Pratt

University of Cambridge Computer Laboratory, Cambridge, UK
{tim.harris,keir.fraser,ian.pratt}@cl.cam.ac.uk

Abstract. Work on non-blocking data structures has proposed extending processor designs with a compare-and-swap primitive, `CAS2`, which acts on two arbitrary memory locations. Experience suggested that current operations, typically single-word compare-and-swap (`CAS1`), are not expressive enough to be used alone in an efficient manner. In this paper we build `CAS2` from `CAS1` and, in fact, build an arbitrary multi-word compare-and-swap (`CASN`). Our design requires only the primitives available on contemporary systems, reserves a small and constant amount of space in each word updated (either 0 or 2 bits) and permits non-overlapping updates to occur concurrently. This provides compelling evidence that current primitives are not only universal in the theoretical sense introduced by Herlihy, but are also universal in their use as foundations for practical algorithms. This provides a straightforward mechanism for deploying many of the interesting non-blocking data structures presented in the literature that have previously required `CAS2`.

1 Introduction

`CASN` is an operation for shared-memory systems that reads the contents of a series of locations, compares these against specified values and, if they all match, updates the locations with a further set of values. All this is performed atomically with respect to other `CASN` operations and specialized reads. The implementation of a non-blocking multi-word compare-and-swap operation has been the focus of many research papers [7, 10, 2, 3, 15, 5]. As we will show none of these provides a solution that is practicable in terms of the operations it requires from the processor, its storage costs and the features it supplies.

This paper presents a new design that solves these problems. The solution is valuable because it finally allows many algorithms requiring `CASN` to be used in earnest (for example those from [11, 5, 4]). `CASN` is useful as a foundation for building concurrent data structures because it can update a set of locations between consistent states. Aside from its applicability, our solution is notable in that it considers the full implementation path of the algorithm. Previous work has often needed a series of abstractions to build strong primitives from those actually available – each layer adds costs, the sum of which places the algorithm beyond reasonable use.