J. Parallel Distrib. Comput. 113 (2018) 250–260

Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

journal homepage: www.elsevier.com/locate/jpdc

Concurrent use of write-once memory☆

James Aspnes [a], Keren Censor-Hillel [b,*], Eitan Yaakobi [b]

[a] Yale University, Department of Computer Science, United States
[b] Technion, Department of Computer Science, Israel

HIGHLIGHTS

- From write-once bits to write-once registers.
- Atomic multi-bit writes.
- Registers based on the tabular WOM code.
- An unrestricted MWMR implementation based on max registers.
- Lower bounds.

ARTICLE INFO

Article history:
Received 12 January 2017
Received in revised form 11 September 2017
Accepted 5 December 2017
Available online 15 December 2017

Keywords:
Concurrent algorithms
Write-once memory
Space complexity

ABSTRACT

We consider the problem of implementing general shared-memory objects on top of write-once bits, which can be changed from 0 to 1 but not back again. In a sequential setting, write-once memory (WOM) codes have been developed that allow simulating memory that support multiple writes, even of large values, setting an average of $1 + o(1)$ write-once bits per write. We show that similar space efficiencies can be obtained in a concurrent setting, though at the cost of high time complexity and fixed bound on the number of write operations. As an alternative, we give an implementation that permits unboundedly many writes and has much better amortized time complexity, but at the cost of unbounded space complexity. Whether one can obtain both low time complexity and low space complexity in the same implementation remains open.

1. Introduction

Write-once memory (WOM) is a storage medium with memory elements, called cells, that can only *increase* their value. These media can be represented as a collection of binary cells, each of which initially represents a bit value 0 that can be *irreversibly* overwritten with a bit value 1. WOM codes, first introduced by Rivest and Shamir [33], enable to record data multiple times without violating the asymmetry writing constraint in a WOM. The goal in the design of a WOM code is to maximize the total number of bits that can be written to the memory in t writes, while preserving the property that cells can only increase their level.

These codes were first motivated by storage media such as punch cards and optical storage. However, in the last decade, a wide study of these codes re-emerged due to their connection to *Flash memories*. Flash memories contain floating gate cells which are electrically charged with electrons to represent the cell level. While it is fast and simple to increase a cell level, reducing its level requires a long and cumbersome operation of first erasing its entire containing block and only then programming (increasing the level of) relevant cells, which are the ones that we need to remain representing a 1 bit. Applying a WOM code enables additional writes before having to physically erase the entire block.

This paper provides the first study of concurrency in write-once shared memory. We investigate concurrent write-once memory from a theoretical viewpoint, which, in particular, means that we consider the memory impossible to erase (as opposed to considering it to be expensive). We show that any problem that can be solved in a standard shared-memory model can be solved in a write-once memory model, at the cost of some overhead. Our goal is to provide an analysis of this cost, both in terms of step complexity and space complexity.

Motivation. In addition to our interest in WOM as a computing model, our study is motivated by two observations. First, WOM is not subject to the **ABA problem**, in which memory can change back and forth going unnoticed, which is proven to be hard to overcome [1].

☆ A preliminary version of this paper appeared in the proceedings of The 23rd International Colloquium on Structural Information and Communication Complexity (SIROCCO), pages 127–142, 2016.

* Corresponding author.
E-mail addresses: aspnes@cs.yale.edu (J. Aspnes), ckeren@cs.technion.ac.il (K. Censor-Hillel), yaakobi@cs.technion.ac.il (E. Yaakobi).

https://doi.org/10.1016/j.jpdc.2017.12.001
0743-7315/

Table 1
A WOM code example.

Data bits	First write	Second write
00	000	111
10	100	011
01	010	101
11	001	110

The second reason is that several known concurrent algorithms are already implemented using write-once bits. In other words, for some specific problems, the overhead of using WOM can be reduced compared to the general case. Examples of such implementations are the **sifters** constructed by Alistarh and Aspnes [2] and by Giakkoupis and Woelfel [17], and some variants of the **conflict detectors**, of Aspnes and Ellen [5].[1] A **max register** [3] is another example of an object that can be implemented using write-once bits (see overview in Section 5). Interestingly, the covering arguments used to prove lower bounds on max registers [3] imply that no historyless primitive can give a better implementation than write-once bits.

Yet these specific solutions do not immediately give a general implementation of arbitrary shared-memory objects, and the question arises whether the space efficiencies obtained by WOM codes in a sequential setting can transfer to a concurrent setting as well.

The challenge. To give a flavor of the challenge in adopting known WOM codes to concurrent use, we explain a simple example in Table 1, introduced by Rivest and Shamir [33], which enables the recording of two bits of information in three cells twice. It is possible to verify that after the first 2-bit data vector is encoded into a 3-bit codeword, if the second 2-bit data vector is different from the first, the 3-bit codeword into which it is encoded does not change any code bit 1 into a code bit 0, ensuring that it can be recorded in the write-once medium.

Suppose now that the above code is used in a concurrent WOM system, and that two processes p_1 and p_2 invoke write operations with input data bits 10 and 01, respectively. This means that p_1 needs to write 100 into the memory, and p_2 needs to write 010 to it. In other words, p_1 needs to set the first of the three bits to 1, while p_2 needs to set the second. Consider a schedule in which p_1, p_2 set their respective bit in some order, and afterwards another process p_3 reads the shared memory. The bits that it sees are 110, but these correspond to the input 11, which was never written into the memory, violating the specification of the memory.

The difficulty above is amplified by the fact that since more than a single process is writing and reading the content of the memory, it is not known what the value of t is, that is, how many writes have occurred so far. This is needed in the above example for both writing and reading.

We emphasize that there is a significant amount of fundamental simulations of different types of registers in the literature of distributed computing (see, e.g., [7, Chapter 10] and [21, Chapter 4]). The above WOM example satisfies the definition of a **single-writer–multi-reader (SWMR) safe register** [28,29], in which a read that is not concurrent with a write returns a correct value. Known simulations can use this object to construct **multi-writer–multi-reader (MWMR) atomic registers**. However, these simulations do not comply with the restrictions of not being able to overwrite a 1 bit with a 0 bit, which arise from WOM, and hence different solutions must be sought.

[1] This does not include the $\Theta(\log m/\log\log m)$-step m-valued conflict detector that appears in [5], but does include a simpler $\Theta(\log m)$-step conflict detector in which a write of a value whose bits are $x_{k-1}, \ldots, x_0$ is done by setting to 1 the corresponding bits $A[i][x_i]$ in a $k \times 2$ array A.

1.1. Our contribution

We first show that with one additional bit that indicates to read operations that a write operation has been completed, we can easily implement a write-once m-bit register. Then, we show how to support t writes, still for a single writer, within a space complexity of $2m + t$ bits. After these toy examples, our goal is to get closer to the $t(1 + o(1))$-space WOM code constructions for the non-concurrent setting. Carefully adapting the tabular code of [33] to our concurrent setting, allows us to obtain a SWMR m-bit register that supports t writes, with the following properties.

Theorem 4.1. *There is an algorithm that implements an n-process SWMR m-bit register supporting up to t writes, using space complexity of $(1+o(1))t$ when $t = \omega(m2^m)$, and with amortized step complexity $O(n2^m)$ for a write and $O(2^m)$ for a read.*

We then extend our tabular construction to support multiple writers, with the aid of a reduction from MWMR registers to SWMR registers due to [24], and with incorporating safe-agreement objects [10] in order to efficiently share space. Our result is summarized as follows.

Theorem 4.2. *There is an algorithm that implements an n-process MWMR m-bit register that supports up to t writes, using space complexity of $(2 + o(1))t$ when $t = \omega((m + \log n)n^6 2^m)$, and with amortized step complexity $O(n^2 2^m)$ for both write and read operations.*

The drawback of the above implementation is its large step complexity. At the cost of increased space complexity, we show how to build a WOM code on top of a max register, which allows drastically reduced step complexities, as stated next. Here, a unique timestamp t is guaranteed to be associated by the algorithm with each write operation.

Theorem 5.1. *There is an algorithm that implements an n-process MWMR register of m bits with unbounded space, where the* amortized *step complexity of a write operation that gets associated with a timestamp t is $O(\log t + m + \log n)$ and the step complexity of a read operation that reads a value associated with a timestamp t is $O(\log t + m + \log n)$.*

Whether it is possible to obtain both low time complexity and low space complexity in the same implementation remains an intriguing open question.

1.2. Additional related work

In their pioneering work, Rivest and Shamir also reported on more WOM code constructions, including tabular WOM codes and linear WOM codes. Since then, several more constructions were studied in the 1980s and 1990s [13,15,18], and more interest to these codes was given in the past seven years; see e.g. [9,11,12,14, 34,35,37–40]. The capacity of a WOM was also rigorously investigated. The maximum sum-rate as well as the capacity regions were studied in [19,33,36] with extensions to the non-binary case in [16]. The implementation of WOM codes in several applications such flash memories and phase-change memories was recently explored in [26,30,31,41,42]. These works were motivated by the system implementation on WOM codes in these memories, while taking into account the hardware and architecture limitations when implementing these codes into the system.

Write-once memory should not be confused with **sticky registers** as defined by Plotkin [32], which in some recent systems literature (e.g. [8]) have been described as registers with **write-once semantics**. Sticky registers initially hold a default "empty" value, and any write after the first has no effect. Such registers are equivalent to consensus objects, and thus significantly more powerful than standard shared memory. In contrast, write-once memory as considered here and in the WOM code literature is weaker than standard shared memory.