CCIT REPORT #765 May 2010

On Maintaining Multiple Versions in STM

Dmitri Perelman* Rui Fan* Idit Keidar*

May 16, 2010

Abstract

An effective way to reduce the number of aborts in software transactional memory (STM) is to keep multiple versions of transactional objects. In this paper, we study inherent properties of STMs that use multiple versions to guarantee successful commits of all read-only transactions.

We first show that these STMs cannot be disjoint-access parallel. We then consider the problem of garbage collecting old object versions, and show that no STM can be optimal in the number of previous versions kept. Moreover, we show that garbage collecting useless versions is impossible in STMs that implement invisible reads. Finally, we present an STM algorithm using visible reads that efficiently garbage collects useless object versions.

1 Introduction

Transactional memory [12, 18] is a popular paradigm for concurrent computing in modern multi-core architectures. Most current transactional memory implementations are software toolkits, or *STM*s for short. STMs speculatively allow multiple transactions to proceed concurrently, before knowing all possible data dependencies between them. This optimistic approach inevitably leads to aborting transactions in some cases, such as when data dependencies introduce inconsistencies. When many transactions contend on the same data objects, aborts may become frequent, causing a devastating effect on performance [2, 15]. Therefore, reducing the number of aborts is an important challenge for STMs.

While some aborts are unavoidable, existing STMs tend to be over-conservative, and also abort transactions that could have been committed without violating consistency. Such unnecessary aborts often stem from coarse-grained inconsistency detection. Consider the scenario depicted in Figure 1. We depict transactional histories in the style of [17]. An object o_i's state in time is represented as a horizontal line, with time proceeding left to right. Transactions are drawn as polylines, with circles representing accesses to objects. Filled circles indicate writes, and empty circles indicate reads. A commit is indicated by the letter **C**, and an abort by the letter **A**. A read operation returning an old value of an object is indicated by a dotted arc line. The initial value of object o_i is denoted by o_i^0, and the value written to o_i by the j'th write is denoted by o_i^j. In the scenario depicted in Figure 1 transaction T_2 reads an object o_1, then another transaction T_3 updates objects o_1 and o_2, and commits. Assume that T_2 now tries to read o_2. Reading the value o_2^2 written by T_3 would violate correctness, since T_2 does not read the value o_2^1 written by T_3. In a single-versioned STM, illustrated in Figure 1(a), T_2 must abort. However, a multi-versioned STM may keep both versions o_2^1 and o_2^2 of o_2, and may return o_2^1 to T_2, as illustrated in Figure 1(b). This allows T_2 to successfully commit, in spite of its conflict with T_3.

*Department of Electrical Engineering, Technion, Haifa, Israel {dima39@tx,rfan@ee,idish@ee}.technion.ac.il