

Software Transactional Objects

(position paper for TRAMP 07)

Guy Eddon Maurice Herlihy

Computer Science Department
Brown University

mph@cs.brown.edu

Introduction

Much current work on language support for transactions has focused on unmanaged languages such as C or C++. Nevertheless, support for managed languages¹ such as Java or C# will be just as important in the long run. Moreover, managed languages make different demands and benefit from different techniques than unmanaged languages.

We are exploring an approach we call *Software Transactional Objects*. In this model, programmers declare certain objects to be *atomic*, and the underlying STM system interposes synchronization and recovery code at each field access. From a software engineering point of view, this approach is attractive for several reasons.

- Programmers continue to program in a mostly sequential style
- Synchronization conflicts between transactional and non-transactional threads can be detected and handled in a natural and efficient way (the so-called *strong atomicity* problem).
- There is a clean separation between the underlying STM machinery and the application code.

We have been involved in two STM libraries based on this approach: SXM uses C# and .Net, and DSTM2 uses Java™.

There is, however, a major drawback to this approach. Synchronizing at the granularity of field accesses can be expensive. Naturally, this overhead could be reduced by synchronizing at a coarser granularity (as in DSTM) by explicitly opening and closing atomic objects, but the resulting conventions are often unwieldy and error-prone.

Instead, we are exploring the use of simple compiler flow analysis to enhance the efficiency of field-level synchronization. We considered different STM mechanisms: the *TMem* atomic object factory uses shadow copies for recovery and short critical sections (intended to mimic limited hardware transactional memory), and *OFree* is an obstruction-free implementation that uses *compare-and-swap* calls for short-term synchronization.

¹ Managed languages typically provide garbage collection, and array bounds checks, and prohibit pointer arithmetic.

TMem								
Benchmark	SXM 1.1	Library Opts	Compiler	Const	Local	RWPromo	Subsequent	PreOpen
List	102,000	226,000	619,000	622,000	626,000	632,000	626,000	849,000
RBTree	64,500	141,000	366,000	380,000	382,000	385,000	392,000	670,000
SkipList	31,000	58,000	133,000	162,000	164,000	155,000	156,000	164,000
HashTable	71,000	293,000	902,000	902,000	898,000	916,000	963,000	1,020,000
Buffer	196,000	415,000	1,114,000	1,223,000	1,228,000	1,292,000	1,294,000	1,363,000

OFree								
Benchmark	SXM 1.1	Library Opts	Compiler	Const	Local	RWPromo	Subsequent	PreOpen
List	10,300	226,000	499,000	500,000	512,000	512,000	507,000	564,000
RBTree	600	150,000	333,000	334,000	330,000	328,000	319,000	372,000
SkipList	1,000	62,000	144,000	135,000	136,000	145,000	148,000	128,000
HashTable	47,000	303,000	801,000	801,000	799,000	810,000	822,000	825,000
Buffer	156,000	536,000	899,000	934,000	918,000	993,000	996,000	970,000

Figure 1: Effect of various optimizations on the SXM 1.1 Library

The first column shows the performance of various benchmarks using the library directly. Each column shows the effects of successive compiler-based optimizations. These optimizations include

- Library Opts - Optimizations to the SXM
- Compiler - Basic bytecode rewriting compiler
- Const - Skip STM calls for const & readonly fields
- Local - Detect and skip STM calls for locally created objects
- RWPromo - Promote OpenReads to OpenWrite for objects that are read and written
- Subsequent - Inline fast-path code for subsequent accesses
- PreOpen - Partial redundancy elimination (PRE) with early acquisition of objects

Before the optimizations, the obstruction-free implementation was an order of magnitude slower than the lock-based implementation, but after the optimizations, they are substantially closer, although the lock-based implementation is still more efficient.

We think that the combination of software transactional objects and compiler optimization is one of the more promising approaches to providing transactional support for managed languages.

Bibliography

Herlihy, M., Luchangco, V., and Moir, M. 2006. A flexible framework for implementing software transactional memory. In *Proceedings of the 21st Annual ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages, and Applications* (Portland, Oregon, USA, October 22 - 26, 2006). OOPSLA '06. ACM Press, New York, NY, 253-262. DOI= <http://doi.acm.org/10.1145/1167473.1167495>

C# Software Transactional Memory,
 javascript: function%20openEula(url)%7bwindow.open(url,'eula','width=500,height=500,

resizable=yes,scrollbars=yes,address=no,menu=no');%7dopenEula('http://research.microsoft.com/research/downloads/download.aspx?FUID=%7bFBE1CF9A-C6AC-4BBB-B5E9-D1FDA49ECAD9%7d');