Brief Announcement: COP Composition Using Transaction Suspension in the Compiler

Hillel Avni[1] and Adi Suissa-Peleg[2]

[1] Tel-Aviv University
hillel.avni@gmail.com
[2] Harvard University
adisuis@seas.harvard.edu

Abstract. Combining a number of transactions into a single atomic transaction is an important transactional memory (TM) feature supported by many software TM (STM) implementations. This composition, however, typically results in long transactions with an increased contention probability.
In consistency oblivious programming (COP), the read-only prefix of a data structure operation is performed outside of a TM transactional context. The operation is then completed by using a transaction that verifies the prefix output and performs updates. In STM, this strategy effectively reduces much of the overhead and potential contention.
In this work we emphasize the importance of *transaction-suspension*, which enables performing non-transactional memory accesses inside a transaction. Suspension not only simplifies the use of COP, but also enables the composition of a sequence of COP-based operations into a single transaction. We add transaction-suspension support to GCC-TM, and integrate COP into TM applications. We also support TM-Safe memory reclamation in transactions with COP operations, by adding privatization before a transaction abort to the GCC-TM library.

Introduction Consistency Oblivious Programming (COP) [2], is a programming methodology for improving a TM-based data structure performance. In COP, the read-only prefix (ROP) of a data structure operation is performed in a non-transactional context. The operation is then completed by using a transaction that verifies the ROP output and performs updates. COP-based data structures effectively reduce much of the TM instrumentation overhead and potential contention.

The ROP may observe inconsistent states, and must avoid crashing as a result. It is the responsibility of the programmer to keep the ROP from hitting infinite loops or uninitialized pointers. Another type of crash may be caused by an ROP code segment that accesses a memory location after it was released by a concurrent transaction. To prevent this scenario we modified the privatization algorithm in the STM.

A useful feature supported by many STM implementations is transactions composability, the ability to combine a number of transactional atomic blocks to be executed in a single transaction. This fosters the use of TM-based data structures, and facilitates the creation of non-trivial atomic transactions that access different data structures.

In this paper, we introduce a methodology that uses GCC-TM, the GNU C Compiler (GCC) [1] STM implementation, to support efficient and natural composition of COP operations. Our methodology is based on *transaction-suspension*, which enables executing non-transactional, non-instrumented instructions inside a transactional block. In order to support a suspension of a transaction in GCC-TM, we mark functions with the TM-Pure attribute[3] [4], that omits the instrumentation of these functions when called from transactions. We apply our methodology to the linked

[3] A function that is marked with the TM-Pure attribute is executed as a non-transactional code block. The TM-Pure attribute is supported by the GCC-TM implementation.

list and red-black tree, that are part of the data-structures library which is used by the STAMP applications. Our results show that this mechanism reduces 80% of the aborts caused by conflicts.

COP Composing using Suspended Transactions When using transaction-suspension, a COP operation, OP, embedded in a transaction T, goes through the following steps:

$\mathrm{T}_{start} \rightarrow$Any code$\rightarrow T_{suspend} \rightarrow \mathrm{OP}_{rop} \rightarrow \mathrm{T}_{resume} \rightarrow \mathrm{OP}_{verify} \rightarrow \mathrm{OP}_{updates} \rightarrow$Any code$\rightarrow \mathrm{T}_{end}$

OP_{verify} should verify the validity of the data gathered during the ROP code. This code is executed locally and must be concise, so that it does not introduce additional overhead.

In addition, note that OP_{rop} can be executed several times in non-transactional, suspended mode, and only if verification failure persists, it should fallback to transactional mode. If the transactional execution of the ROP, i.e., the fallback, aborts, the transaction naturally aborts.

The only way to compose COP operations without transaction-suspension, is the one proposed by [5], i.e., execute all ROP parts of the composed operations before starting the transaction, then, inside the transaction, verify their output and complete the transactions updates. This method allows composition only if an operation is not writing data that may later be accessed by another COP operation in the same transaction.

Safe Memory Reclamation Two important functions that are TM-Safe [4], i.e., can be executed inside a transaction, are *malloc* and *free*. These functions are made safe by privatization. If transaction T wrote to memory, then before it commits, it waits for the termination of the transactions that started before its commit [3]. As a side effect of privatization, in case T detaches some memory block from a data structure and successfully commits, then T can free that block.

On the other hand, if T allocates some block of memory, M, and then aborts, it can free M without privatization. The reason is that the pointer to the tentative memory block is not exposed to other transactions.

This is violated when COP is involved. If the non-transactional ROP code block traverses the data structure, it may acquire a pointer to a newly allocated memory block, and upon an abort of T and freeing M, the ROP may try to access unmapped memory. To prevent this scenario, we added privatization to writing transactions that are about to perform rollback. If a transaction is a read-only transaction, it can free its tentative memory blocks unconditionally. If, however, the transaction updated some memory location, it has to perform privatization as if it was successfully committed. Our evaluation showed that this privatization has a negligible impact on performance.

With the suspended mode, and rollback privatization, malloc and free become also COP safe. The reason is that memory is not recycled as long as there is a transaction in progress, and the COP operations are always encapsulated in transactions. One restriction is that allocation cannot take place in a ROP, because, in case the validation fails, the allocated memory will not be freed, as we do not abort the transaction in this case. However, as the ROP code typically avoids from writing to memory, it does not need to allocate or free memory.

References

1. Gcc version 4.7.0, (http://gcc.gnu.org/gcc-4.7/), Apr. 2012.
2. Y. Afek, H. Avni, and N. Shavit. Towards consistency oblivious programming. In *OPODIS*, pages 65–79, 2011.
3. D. Dice, A. Matveev, and N. Shavit. Implicit privatization using private transactions. In *TRANSACT*, 2010.
4. T. Riegel. *Software Transactional Memory Building Blocks.* PhD thesis, Technischen Universitat Dresden, geboren am 1.3.1979 in Dresden, 3 2013.
5. L. Xiang and M. L. Scott. Composable partitioned transactions. In *WTTM*, 2013.