TM-Pure in GCC Compiler Allows Consistency Oblivious Composition

Hillel Avni
Tel-Aviv University
hillel.avni@gmail.com

Adi Suissa
Ben-Gurion University
adisuis@cs.bgu.ac.il

Abstract

In consistency oblivious programming (COP), we leave the read-only prefix of a data structure operation, out of the TM transaction. Then we complete the operation in a transaction by verifying the prefix output, and performing updates. In STM, this removes a large part of the overhead, and potential contention.

If a TM transaction can not be suspended, as the case for example with Haswell HTM block, when we want to concatenate several COP operations in a single transaction, we have to run all the read-only prefixes separately before the transaction, and log their outputs, and then start a transaction to validate all outputs and complete the operations. This format does not allow a transaction to update a data structure and later access it, as it will miss its own updates. Also, if the transaction started with an updating operation, such as a dequeue, it can not benefit from COP anymore.

We use TM-Pure mode, supported by GCC, to concatenate COP operations which access a data structure that was modified earlier by the transaction. As Haswell HTM transaction can not be suspended, GCC with TM-Pure mode, is demonstrating much better performance than HTM for many workloads, by composing COP operations.

Keywords Transactional-Memory, Consistency Oblivious Programming, Data-Structures

1. STM in GCC Compiler

In GCC STM support, the transaction code is marked by the $\_\_transaction\_atomic\{\}$ directive. Inside the transaction, all accesses to shared memory are instrumented, which means instead of plan load or store machine instruction, a library function is called. The function can lock the location for writing, log it for later verification or rollback, or check the version it, according to the STM algorithm implemented in the library.

The STM libraries are divided by a set of typical characteristics. A library is either write through, which means it may write the newly stored value to the destination address in memory, or write back, so the value is kept in a redo log for commit time, visible only locally. It can be serial, i.e., just take a global lock, or have multi-locks, which means it maps a lock per set of addresses, like a set of addresses may map to an index in the cache. Once the library executes a transactional store, it may lock the address immediately, in encounter time, or just in commit time. If the library is write-through, it must be encounter-time-locking, to maintain isolation. TM research proves that the most efficient and scalable STM is multi-lock, encounter-time and write-through [4, 5], and thus, this is the default algorithm in GCC STM library.

During compilation, The compiler infers what addresses in the transaction can be shared and plants a call to a store or load function from its library instead of the plan machine instruction.

In GCC, the transaction code can call functions that are attributed by either $\_\_transaction\_safe$ or $\_\_transaction\_pure$ [7]. $\_\_transaction\_safe$ functions are transactionally instrumented, but the $\_\_transaction\_pure$ functions are kept unchanged.

The philosophy behind STM is that the programmer can be trusted, and can make the decision if a segment of code does not require instrumentation. As STM, unlike TM in hardware, is not aware of any non transactional access to memory, it is assumed such races are in the jurisdiction of the user. The $\_\_transaction\_pure$ is not changing this philosophy.

A transaction can call $\_\_transaction\_cancel$, which will abort the transaction, i.e., discard and undo all its updates, and rollback the program as if the transactions never happened. The STM algorithm is best effort, in the sense that if it does not manage to commit successfully in a certain number of retries, it takes a global lock and executes the code pessimistically. In serial fall back mode, a transaction must not call call $\_\_transaction\_cancel$, otherwise it may hit a live lock.

2. TM-Pure and Suspended Semantics

The TM-Pure functions, with the default write-through library, has surprisingly similar semantics to future POWER architecture HTM block [2] suspended state, which is marked by the newly introduced instructions `tsuspend` and `tresume`. We will follow the POWER semantics and see how they are correlated in TM-Pure code of GCC STM:

SEMANTIC 1. *Until failure occurs, load instructions that access memory locations that were transactionally written by the same thread will return the transactionally written data.*

As the address was written transactionally by GCC, it is locked, and the value is in memory. This implies that a load instruction from an address that was previously written by the transaction, will return the transactionally written value.

SEMANTIC 2. *In the event of transaction failure, failure recording is performed, but failure handling is deferred until transactional execution is resumed.*

If our STM transaction got intersected by another transaction, it will definitely fail when we return from the TM-Pure code.

SEMANTIC 3. *The initiation of a new transaction is prevented.*

This can be prevented in compile time, but currently we trust the user for it.

SEMANTIC 4. *Store instructions that access memory locations that have been accessed transactionally (due to load or store) by the same thread will cause the transaction to fail.*