Exercises on Transactional Memory

Maurice Herlihy

June 19, 2017

Question

In the hardware transactional memory design shown in the lecture, a processor that writes to a memory location invalidates all cached copies held by other processes. What are some negative implications of this design?

Question

Define a variable to be *thread local* if (1) each thread has a private copy of the variable, and (2) the variable keeps its value from one teleport call to the next.

Recall that in the lock teleportation pseudocode, the thread-local teleportLimit variable controls how many nodes the teleport transaction traverses.

In the teleportation shown in the lecture, the starting node is unlocked, and the ending node is locked inside the transaction, before it commits.

Question: Does the order in which the locks are acquired and released matter?

Question: Would it be correct to acquire and release those locks after the transaction has committed?

Question

The teleportLimit variable is incremented by one after a transaction successfully commits. Would it be correct to increment that variable inside the transaction? If not, explain why. If so, explain why the increment is performed outside the transaction even if it is not required for correctness.

Question

If a transaction aborts, the `_xbegin()` call returns a code explaining why the transaction aborted. Here are some possibilities:

- there was a synchronization conflict with another transaction
- the transaction overflowed the cache
- the transaction executed an illegal instruction
- the transaction was aborted by a timer interrupt
- unknown reason

The teleportation pseudocode ignores the reason for a transaction abort: it simply retries for a fixed number of times. Show how the teleportation code could be more effective by taking the abort code into account.

Question

In the first STM algorithm described in the lecture, a reader checks whether a memory location is locked before reading. Give an example showing why this is necessary.

Question

In that STM algorithm, when we commit a transaction, we lock all locations that were written. Is there a danger here? Explain how to avoid it.

Question

Does transactional memory have a problem with I/O? Explain the nature of the problem, and suggest some partial solutions.

Question

Deadlock occurs when a set of threads are blocked, each waiting for another to do something. *Livelock* occurs when a set of speculative activities repeated cause one another to restart. Both deadlock and livelock can prevent progress.

Is the hardware transactional memory design described in the notes subject to deadlock? How about Livelock?