Brief Announcement: Faster Data Structures in Transactional Memory using Three Paths

Trevor Brown [⋆]

University of Toronto, Toronto, Ontario, Canada

With the introduction of Intel's restricted hardware transactional memory (HTM) in commodity hardware, the transactional memory abstraction has finally become practical to use. Transactional memory allows a programmer to easily implement safe concurrent code by specifying that certain blocks of code should be executed atomically. However, Intel's HTM implementation does not offer any progress guarantees. Even in a single threaded system, a transaction can repeatedly fail for complex reasons. Consequently, any code that uses HTM must also provide a non-transactional fallback path to be executed if a transaction fails. Since the primary goal of HTM is to simplify the task of writing concurrent code, a typical fallback path simply acquires a global lock, and then runs the same code as the transaction. This is essentially transactional lock elision (TLE). Changes made by a process on the fallback path are not atomic, so transactions that run concurrently with a process on the fallback path may see inconsistent state. Thus, at the beginning of each transaction, a process reads the state of the global lock and aborts the transaction if it is held.

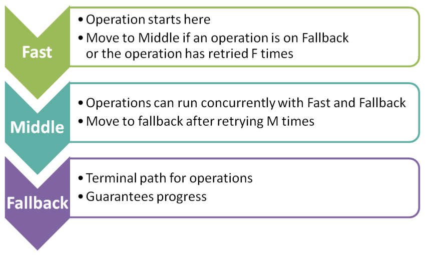
Despite its widespread use, there are many problems with this fallback path. If transactions abort infrequently, then processes rarely execute on the fallback path. However, once one process begins executing on the fallback path, all concurrent transactions will abort, and processes on the fast path will cascade onto the fallback path. This has been called the *lemming effect*, from the myth that lemmings will leap from cliffs in large numbers.

One simple way to mitigate the lemming effect is to retry aborted transactions a few times, waiting between retries for the fallback path to become empty. For some common workloads (e.g., range queries and updates on an ordered set implemented with a binary search tree), some operation is nearly always on the fallback path, so concurrency is very limited and performance is poor. Thus, waiting for the fallback path to become empty is not always a good solution.

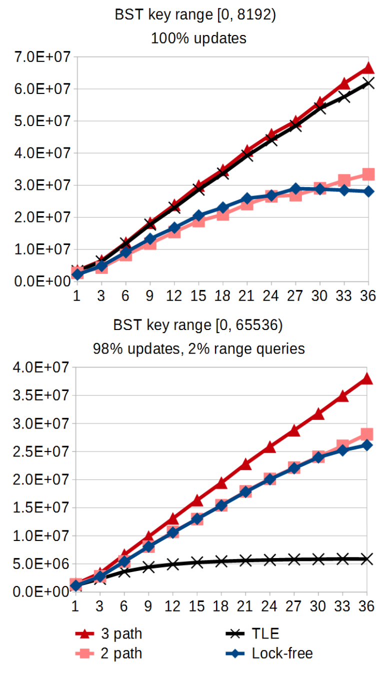
A more sophisticated solution is to design transactions so they can commit even if processes are executing on the fallback path. One way to do this is to start with a hand-crafted fallback path that uses fine-grained synchronization, and obtain a fast path by wrapping each operation in a transaction (and then optimizing the resulting sequential code). This technique was explored by Liu et al. [1]. To support concurrency between the two paths, the fast path must read and update the meta-data used by the fallback path to synchronize pro-

[⋆] This work was supported by NSERC. I thank my advisor Faith Ellen for her helpful comments on this work. Some experiments were performed while at Oracle Labs.

cesses. Unfortunately, the overhead of manipulating meta-data on the fast path can eliminate much or all of the performance benefit of HTM.


Fast
• Operation starts here
• Move to Middle if an operation is on Fallback or the operation has retried F times
Middle
• Operations can run concurrently with Fast and Fallback
• Move to fallback after retrying M times
Fallback
• Terminal path for operations
• Guarantees progress

To overcome this, we introduce a novel approach for obtaining faster algorithms by using three execution paths: an HTM-based fast path, an HTM-based middle path and a non-transactional fallback path. Our approach eliminates the lemming effect without imposing any overhead on the fast path. Each operation begins on the fast path, and moves to the middle path after it retries F times. An operation on the middle path moves to the fallback path after retrying M times on the middle path. The fast path does not manipulate any synchronization meta-data used by the fallback path, so operations on the fast path and fallback path cannot run concurrently. Thus, whenever an operation is on the fallback path, all operations on the fast path move to the middle path. The middle path manipulates the synchronization meta-data used by the fallback path, so operations on the middle path and fallback path can run concurrently. Operations on the middle path can also run concurrently with operations on the fast path. The lemming effect does not occur, since an operation does not have to move to the fallback path simply because another operation is on the fallback path. Since processes on the fast path do not run concurrently with processes on the fallback path, the fallback path does not impose any overhead on the fast path.



Experiments were performed on a 36-core Intel system, comparing the performance of a binary search tree with several two- and three-path algorithms, and different retry strategies, over a variety of workloads. In the 100% update workload, the three-path algorithm matches the performance of TLE and significantly outperforms the other algorithms. In the workload with range queries, TLE succumbs to the lemming effect and performs very poorly. These results suggest that three-path algorithms can be used to obtain the full performance benefit of HTM while robustly avoiding the lemming effect.

References

1. Y. Liu, T. Zhou, and M. Spear. Transactional acceleration of concurrent data structures. In *Proc. of 27th ACM Sym. on Parallelism in Algorithms and Arch.*, SPAA '15, pages 244–253, New York, NY, USA, 2015. ACM.