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

Trevor Brown

▶ **To cite this version:**

Trevor Brown. Brief Announcement: Faster Data Structures in Transactional Memory using Three Paths. DISC 2015, Toshimitsu Masuzawa; Koichi Wada, Oct 2015, Tokyo, Japan. hal-01207905

HAL Id: hal-01207905
https://hal.science/hal-01207905v1

Submitted on 1 Oct 2015

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

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

Trevor Brown *
University of Toronto, Toronto, ON M5S 3G4, Canada

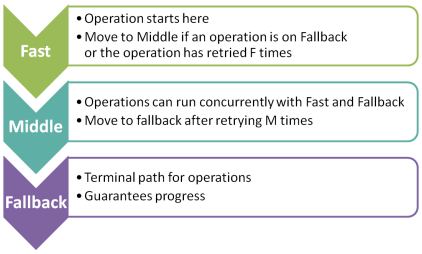
With the introduction of Intel's restricted hardware transactional memory (HTM) in commodity hardware, the transactional memory abstraction has finally become practical to use. Transactional memory allows a programmer to easily implement safe concurrent code by specifying that certain blocks of code 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 (and often undocumented) 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* [1], from the myth that lemmings will leap from cliffs in large numbers (in our case, leaping down to the fallback path).

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.

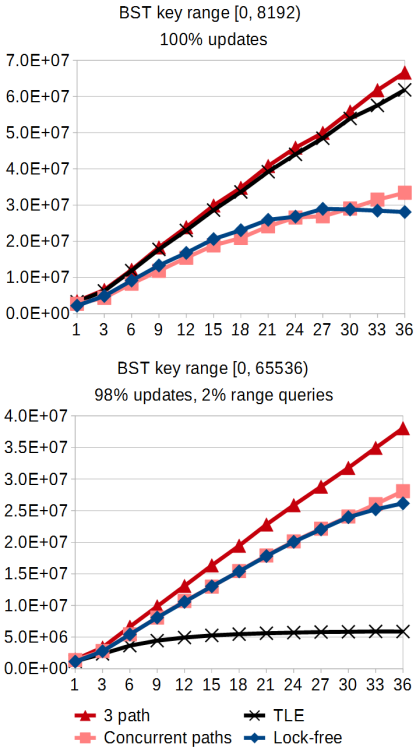
A more sophisticated solution is to design transactions so they can commit even if processes are executing on the fallback path. One way to do this is to start with a hand-crafted fallback path that uses fine-grained synchronization, and obtain a fast path by wrapping each operation in a transaction (and then optimizing the resulting sequential code). This technique was also explored in concurrent work by Spear et al. [2]. To support concurrency between the two paths, the fast path must read and update the meta-data used by the fallback path to synchronize processes. (For example, lock-free algorithms often maintain a record associated with each update operation, so that one process can help complete another process' operation.) Unfortunately, the overhead of manipulating meta-data on the fast path can eliminate much or all of the performance benefit of HTM.

* This work was supported by the National Science and Engineering Research Council of Canada. I extend my thanks to my advisor Faith Ellen for her helpful comments and guidance in shaping this work. Some of the experiments were performed while at Oracle Labs.



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 (regardless of how it operates).

Comprehensive experiments compared the performance of several two- and three-path algorithms, with lock-based and lock-free fallback paths and many different retry strategies, over a variety of randomized workloads. The graphs on the right show some results from a binary search tree microbenchmark on a 36-core Intel system. In the workload with 2% 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. Dave Dice, Yossi Lev, Mark Moir, Dan Nussbaum, and Marek Olszewski. Early experience with a commercial hardware transactional memory implementation. 2009.
2. Yujie Liu, Tingzhe Zhou, and Michael Spear. Transactional acceleration of concurrent data structures. In *Proceedings of the 27th ACM on Symposium on Parallelism in Algorithms and Architectures*, SPAA '15, pages 244–253, New York, NY, USA, 2015. ACM.