

Lock-Aware Transactional Memory

Justin E. Gottschlich¹, Daniel A. Connors², Dwight Y. Winkler³, Jeremy G. Siek¹ and Manish Vachharajani¹

¹Department of Electrical and Computer Engineering, University of Colorado at Boulder

²Department of Electrical and Computer Engineering, Colorado State University

³Nodeka, LLC.

¹{gottschl, jeremy.siek, manishv}@colorado.edu, ²dan.connors@colostate.edu, ³dwright@nodeka.com

Transactional memory (TM) is an emerging concurrency control mechanism that provides a simple and composable programming model. Unfortunately, transactions violate the semantics of mutual exclusion locks when they execute concurrently. Due to the prevalence of locks, transactions must be made *lock-aware* enabling them to correctly interoperate with locks. We present a lock-aware transactional memory (LATM) system that employs a unique communication method using local knowledge of locks coupled with granularity-based policies. Our system allows higher concurrent throughput than prior systems because it only prevents *truly* conflicting critical sections from executing concurrently. Furthermore, our system relaxes the prior requirement of transaction isolation when executing conflicting transactional critical sections and instead runs these transactions as irrevocable, improving transaction concurrency. We demonstrate our performance improvements through empirical benchmarks.

1. Background

When transactions and locks are executed concurrently, they are not guaranteed to behave in a consistent manner due to the differences in their underlying critical section semantics [1, 2]. Mutual exclusion locks use pessimistic critical sections that are limited to one thread of execution. Transactions use optimistic critical sections that support unlimited concurrent thread execution and resolve conflicts that may arise in the transaction’s commit phase.

2. Locks Outside of Transactions (LoT)

Locks outside of transactions, or LoTs, are scenarios where a pessimistic critical section of a lock is executed in one thread while an optimistic critical section of a transaction is concurrently executed in another thread. LoTs require special handling to ensure concurrently executed locks and transactions do not access the same shared memory, as such behavior could result in inconsistencies.

Our largest granularity policy for transaction-lock cooperation, full lock protection, enforces all transactions to commit or abort before a lock’s critical section is executed. TM-lock protection, our mid-level granular lock protection policy, requires some system locking knowledge to use, but achieves increased throughput by reducing some false conflicts found in full lock protection. TX-lock protection, our smallest granularity policy, yields the widest potential concurrency by using specific *local* knowledge of locks, but limits system stalls to only true conflicts. Performance of the three policies populating a linked list is shown in Figure 1.

3. Locks Inside of Transactions (LiT)

Locks inside of transactions, or LiTs, are scenarios where a lock’s pessimistic critical section is executed inside a transaction. Locks placed inside of transactions must behave like normal locks; locking mutual exclusion must be maintained to ensure correctness. In order to support this property and because locks do not have failure atomicity (i.e., they cannot be retried), the transactions containing locks must be run by themselves (e.g., isolated) or run so that they cannot be aborted (e.g., irrevocable).

Our largest granular LiT policy, full lock protection, does not require any local knowledge of locks to execute correctly. It ensures correctness by disallowing the execution of any lock-based or transaction-based critical section while an *isolated* LiT transaction is executing. Our mid-level LiT policy, TM-lock protection, improves on the performance of LiT full lock protection by requir-

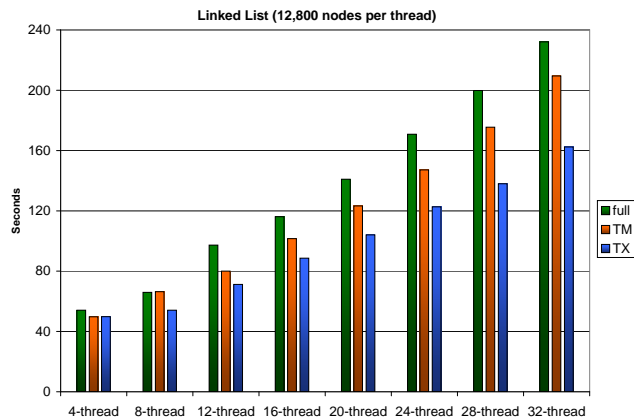


Figure 1. Linked List Benchmark: LoT Lock Protection Policies.

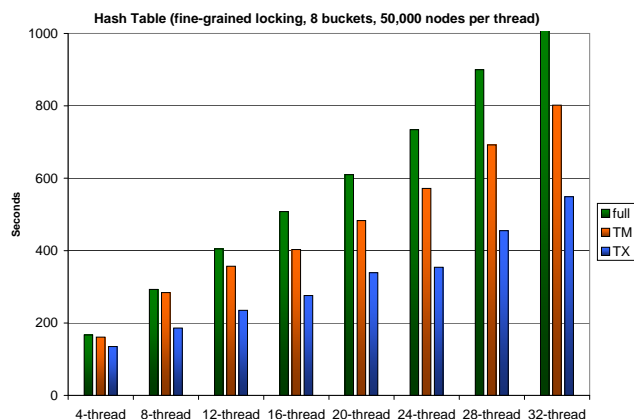


Figure 2. Hash Table Benchmark: LiT Lock Protection Policies.

ing some program-level knowledge of locks which conflict with transactions and allowing non-conflicting locks to execute concurrently with LiT transactions. The most refined policy, LiT TX-lock protection, yields the widest potential concurrency by requiring *local* knowledge of the transactions that conflict with specific locks. Because TX-lock protection is aware of *true* conflicts that exist per transaction, LiT transactions can run as *irrevocable*, rather than isolated, enabling revocable transactions to run alongside it and widening concurrency compared to the other LiT policies and Ziarek et al.’s atomic serialization [2]. Performance of the LiT policies using composed hash table operations are shown in Figure 2.

References

- [1] H. Volos, N. Goyal, and M. M. Swift. Pathological interaction of locks with transactional memory. In *ACM SIGPLAN Workshop on Transactional Computing*, February 2008.
- [2] L. Ziarek, A. Welc, A.-R. Adl-Tabatabai, V. Menon, T. Shpeisman, and S. Jagannathan. A uniform transactional execution environment for Java. In *ECOOP*, pages 129–154, 2008.