poster session, PoPP 2007

Efficient Nonblocking Software Transactional Memory

Virendra J. Marathe
University of Rochester

Mark Moir
Sun Microsystems Laboratories

Transactional Memory (TM)

- A powerful concurrent programming abstraction
- Promises to simplify concurrent programming
- Evolved from earlier work in nonblocking concurrent data structures
- Our work in context of Software TMs (STMs)

The Blocking-Nonblocking Debate

- Recent STM proposals assume blocking systems are inherently faster than nonblocking systems
- Understanding largely based on intuition, and no formal lower bound proofs

Our Argument

- Can build a nonblocking STM that mimics behavior of a fast blocking STM in the common case, resorting to more expensive transactional data displacement only when necessary to guarantee nonblocking progress

Our Idea

- Transaction **steals** ownership of locations if necessary for forward progress
  - Logical contents of stolen locations are displaced to a "different" place
  - All transactions must lookup this alternate location for logical values of a stolen location
  - The system merges logical values in physical locations when no transaction owns the location's ownership record
  - Inspired by Harris and Fraser's stealing methodology

Experimental Setup

- 144-processor SunFire E15K cache coherent multiprocessor with 1.5GHz UltraSPARC® IV+ processors (72 dual core chips)
- Threading levels 1 – 64 (more experiments conducted with up to 256 threads)
- Binary Search Tree (80% lookups, 10% inserts, 10% deletes)

Keys

- Blocking STM
- Nonblocking STM, configured to never steal
- Nonblocking STM
- WSTM (by Harris & Fraser)

Design Details

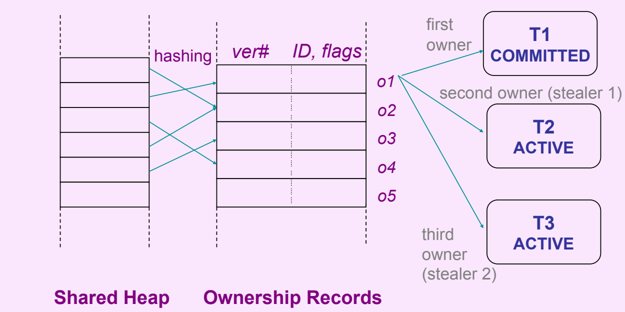
Basic (blocking) Algorithm

- Ownership Record (orec) table
- Each location hashes into one orec
- orec contains owner transaction's ID, version
- Version numbers permit reuse of the same transaction descriptor, and *fast release*
- Transaction contains private read and write sets
- Transaction makes buffered updates (updates are locally maintained in the transaction's write set, and copied back to actual locations on commit)
- Transactions acquire orecs (`CAS` the transaction's ID and version in the orec) of updated locations during the first write
- A transaction blocks when the orec it intends to access is owned by a **COMMITTED** transaction
  - Means that the committed owner is copying back its updates

Extensions for Nonblocking Progress

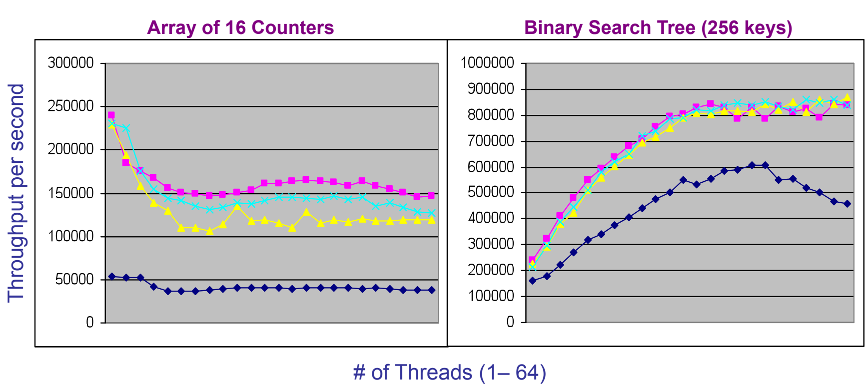
- orec contains a `stolen_orec` flag to identify stolen orecs (logical values of these are displaced in the stealer's descriptor)
- orec contains a `copier_exists` flag to determine that some transaction is merging logical values to physical locations that hash into the stolen orec
- First stealer sets `stolen_orec` and `copier_exists` flags
  - logical values of locations hashing in the stolen orec are in the stealer's descriptor
- Victim resets `copier_exists` flag after its copyback
- A transaction may steal an already stolen orec
- The second stealer checks if `copier_exists` flag is unset
  - if so, sets the flag (while stealing), and assumes the copyback responsibility
  - resets both flags after the copyback if no other transaction stole the orec in the interim
  - means that the logical and physical contents of stolen locations is identical; direct access to locations is permitted

Illustration



Steps

1. T1's copyback in progress; *o1* in unstolen mode, points to T1
2. T2 merges locations in that map into *o1*
3. T2 steals *o1*, setting both flags; logical values are in T2's descriptor
4. T1 finishes copyback, resets `copier_exists` flag
5. T3 decides to steal *o1* from T2, aborts T2, sets `copier_exists` flag, does a copyback, and resets both flags
6. *o1* back in unstolen mode



Conclusions

- Improved significantly over the state-of-the-art nonblocking STM
- Stealing entails noticeable overheads
- Question of inherent cost for providing nonblocking progress remains unclear
- Future Work: Adapt our ideas to other high performance STMs