A Hybrid TM for Haskell[1]

Ryan Yates and Michael L. Scott

University of Rochester

TRANSACT 9, 3-2-2014

[1]This work was supported in part by the National Science Foundation under grants CCR-0963759, CCF-1116055, CNS-1116109, and CCF-1337224.

Outline

- Haskell STM as it is now [Harris et al., 2005].
- Our work, extending Haskell STM into a hybrid TM.
- Haskell-specific challenges and opportunities.
- Future work, supporting more of Haskell TM in hardware transactions.

Haskell STM, The Good

As of writing, around 280 open source libraries in the Haskell ecosystem depend on STM.

Reasons for using Haskell STM:

- It is easy!
- Expressive API with `retry` and `orElse`.
- Trivial to build into libraries.

But. . .

Poor performance.

- The default fine-grain locking scheme performs very poorly under contention.
- Around 7X performance overhead in the best conditions.
- Gets much worse as the number of transactional variables increases.
- Metadata granularity.

Haskell STM Example

```
transA = do
    v <- dequeue queue1
    return v
transB = do
    v <- dequeue queue2
    if someCondition v
      then return v
      else retry
...
mainLoop = do
    a <- atomically $ transA `orElse` transB `orElse` ...
    handleRequest a
    mainLoop
```

Opportunity

Haskell's TM already encourages the use of only a few transactional variables and transactions that are focused on performing a minimal amount of work.

This sounds like a good fit for hardware transactions!

Intel TSX

Transactional Synchronization Extensions (TSX)

- Hardware Lock Elision (HLE)
  - `XACQUIRE` and `XRELEASE` prefixes for lock operations.
- Restricted Transactional Memory (RTM)
  - `XBEGIN` marks beginning of a transaction.
  - `XTEST` determines if execution is in a transaction.
  - `XABORT` aborts a transaction and returns an 8-bit reason code.
  - `XEND` ends a transaction.

Intel TSX

Limitations (4th generation core architecture)

- Cache line granularity.
- Some instructions are not allowed in a transaction.
- Writes limited by L1 data cache capacity and associativity.
- Reads do not appear to have this limitation.
- No guarantee that any particular transaction will complete.

Haskell STM Implementation

- Glasgow Haskell Compiler (GHC), 7.6.3
- Explicit transactional variables.
- Lazy value-based validation.
- Object based.

Haskell STM Implementation

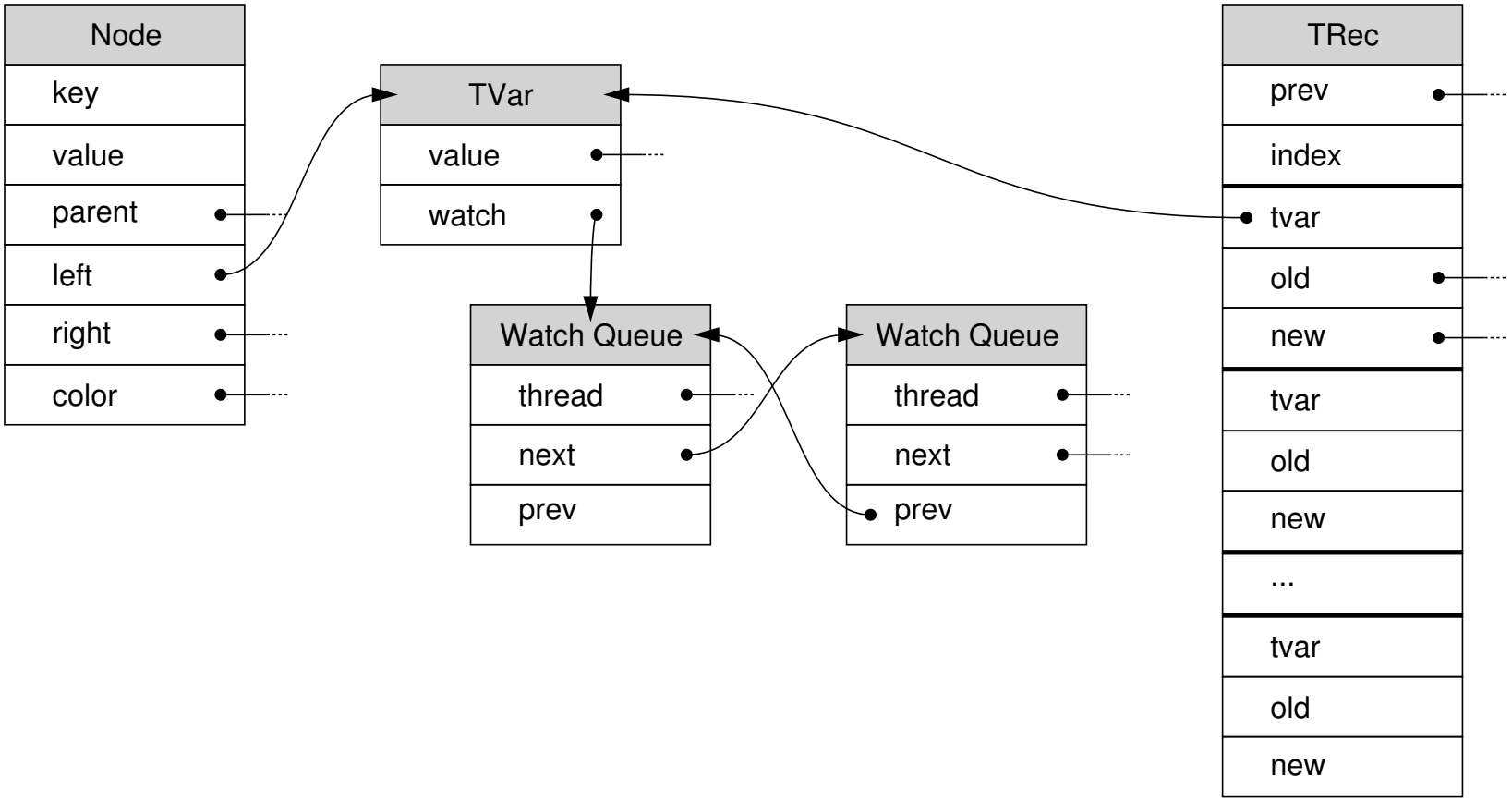
Coarse-grain Lock

- Serialize commits with a global lock
- Similar to NOrec [Dalessandro et al., 2010, Dalessandro et al., 2011, Riegel et al., 2011].

Fine-grain Locks

- Lock for each `TVar`.
- Two-phase commit.
- Similar to object-based STM [Fraser, 2004].

Haskell STM Metadata Structure



Haskell Hybrid TM (Our Work)

- Three levels for transactions [Matveev and Shavit, 2013].
  - Full transactions in hardware.
  - Software transaction, commit in hardware.
  - Full software fallback.

Haskell Hybrid TM (Our Work)

In full hardware transaction we can safely read the global lock just before commit (`XEND`).

- Haskell's limited and explicit effects make this safe.
- If a partial software commit is seen in a hardware transaction, the worst behavior it can do is an infinite loop. This will terminated by GC or context switch.

Implementation Roadblocks

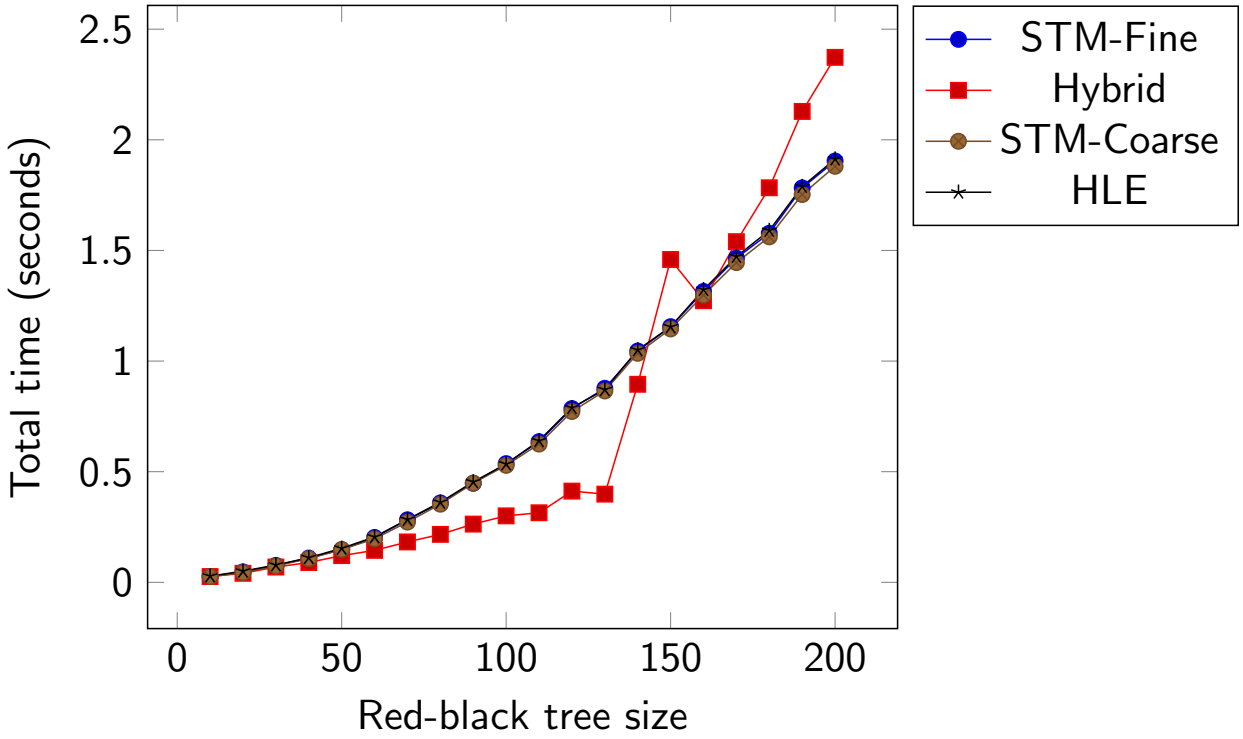
- C ABI vs Cmm ABI
- Indirection overhead

An expected issue for the future

- Lazy evaluation's memoization.

Preliminary Results

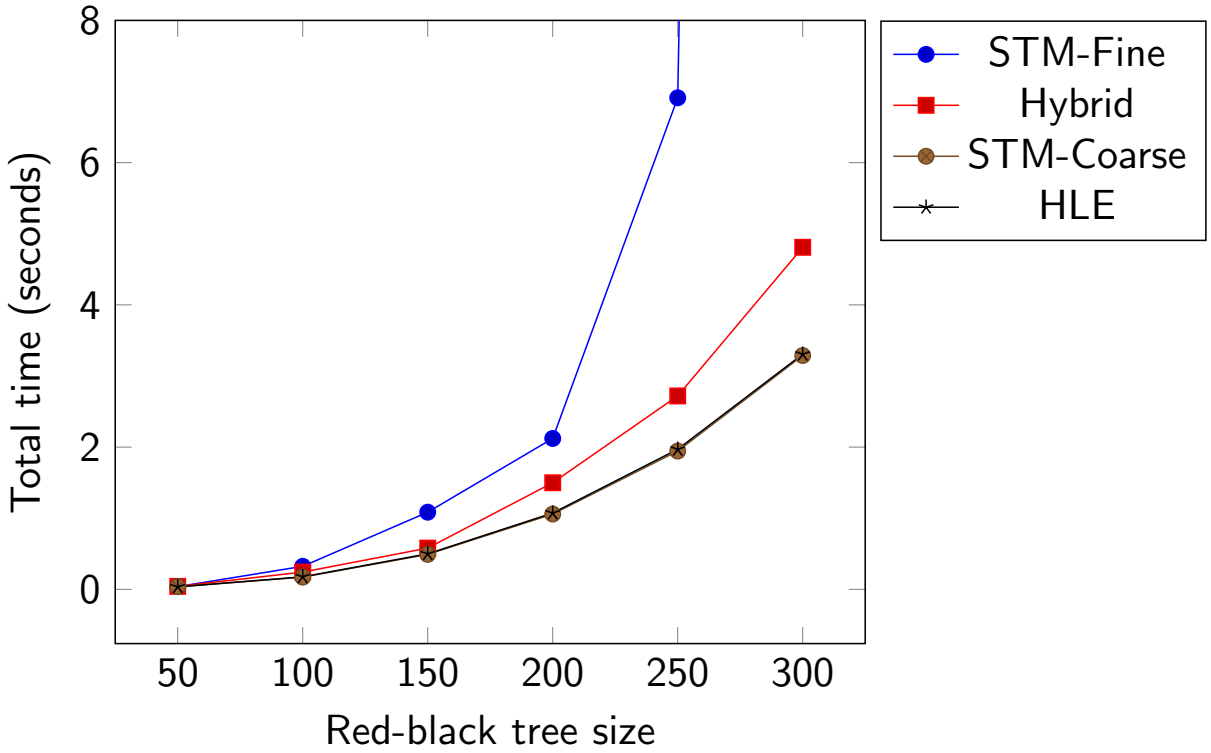
(Intel© Core™ i7-4770, 3.4GHz)



Reads from a red-black tree. At each size tree, each of 4 threads performs 5,000 transactions of 40 lookups each.

Preliminary Results

(Intel© Core™ i7-4770, 3.4GHz)



Writes to a red-black tree. At each size tree, 4 threads together eventually replace every node in the tree 10 times, by atomically deleting one node and inserting another.

Future Implementation Work (`retry`)

Existing `retry` Implementation

- When `retry` is encountered, add the thread to the watch list of of each `TVar` in the transaction's `TRec`.
- When a transaction commits, wake up all transactions in watch lists on `TVars` it writes.

Future Implementation Work (`retry`)

Supporting `retry` in Hardware Transactions

- Throw away the writes *and* record the **read set**.
- `XABORT` only gives us 8-bits. If we had a non-transactional write we could get more.
  - Sacrifice wakeup accuracy with a constant space *approximation* of the read-set.
  - Use the 8-bits as a degenerative Bloom filter.

Future Implementation Work (`orElse`)

Existing `orElse` Implementation

Atomic choice between transactions biased toward the first.

- Nested `TRecs` allow for partial rollback.
- If the first transaction encounters `retry`, throw away the writes, but merge the reads and move to the second transaction.

Future Implementation Work (`orElse`)

Supporting `orElse` in Hardware Transactions

- No direct support in hardware for a partial rollback.
- If the first transaction does not write to any `TVars`, there is nothing to roll back.
  - Keep a `TRec` while running the first transaction.
  - Or rewrite the first transaction to delay writes until after the choice to `retry`.

Summary

- Performance improvements when using HTM.
- Challenges
  - Indirection
  - Calling convention overhead
- Lazy late lock subscription feasible in Haskell.
- False sharing problem induced by lazy evaluation's memoization.
- Plan for handling `retry` and `orElse` in HTM.

Haskell STM TQueue Implementation

```
data TQueue a = TQueue (TVar [a]) (TVar [a])

dequeue :: TQueue a -> a -> STM ()
dequeue (TQueue _ write) v = modifyTVar write (v:)

enqueue :: TQueue a -> STM a
enqueue (TQueue read write) =
    readTVar read >>= \case
        (v:vs) -> writeTVar read vs >> return v
        [] -> reverse <$> readTVar write >>= \case
                []     -> retry
                (v:vs) -> do writeTVar write []
                             writeTVar read vs
                             return v
```

Haskell STM Implementation

Fairly standard commit protocol, but missing optimizations from more recent work.

Commit

- Coarse grain: perform writes while holding the global lock.
- Fine grain:
  - Acquire locks for writes while validating.
  - Check that read-only variables are still valid while holding the write locks.
  - Perform writes and release locks.

Haskell STM

Broken code that we are not allowed to write!

```
transferBad :: TVar Int -> TVar Int -> Int -> STM ()
transferBad accountX accountY value = do
    x <- readTVar accountX
    y <- readTVar accountY

    writeTVar accountX (x + v)
    writeTVar accountY (y - v)

    if x < 0
      then launchMissles
      else return ()
```

Haskell STM

Broken code that we are not allowed to write!

```
thread :: IO ()
thread = do
    transfer a b 200
    transfer a c 300
```

C ABI vs Cmm ABI

- GHC's runtime support for STM is written in C.
- Code is generated in Cmm and calls into the runtime are essentially foreign calls with significant extra overhead.
- We avoid this by writing the HTM support in Cmm.
- Typeclass machinery could allow deeper code specialization.

Lazy Evaluation

- Lazy evaluation may lead to false conflicts due to the update step that writes back the fully evaluated value.
- One solution could be to delay performing updates (to shared values) until after a transaction commits.
- Races here are fine as any update must represent the same value.

References

[Dalessandro et al., 2011] Dalessandro, L., Carouge, F., White, S., Lev, Y., Moir, M., Scott, M. L., and Spear, M. F. (2011).
Hybrid NOrec: A case study in the effectiveness of best effort hardware transactional memory.
In *Proc. of the 16th Intl. Symp. on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 39–52, Newport Beach, CA.

[Dalessandro et al., 2010] Dalessandro, L., Spear, M. F., and Scott, M. L. (2010).
NOrec: Streamlining STM by abolishing ownership records.
In *Proc. of the 15th ACM Symp. on Principles and Practice of Parallel Programming (PPoPP)*, pages 67–78, Bangalore, India.

[Fraser, 2004] Fraser, K. (2004).
Practical lock-freedom.
PhD thesis, University of Cambridge Computer Laboratory.

[Harris et al., 2005] Harris, T., Marlow, S., Peyton Jones, S., and Herlihy, M. (2005).
Composable memory transactions.
In *Proc. of the 10th ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming (PPoPP)*, pages 48–60, Chicago, IL.

[Matveev and Shavit, 2013] Matveev, A. and Shavit, N. (2013).
Reduced hardware NOrec.
In *5th Workshop on the Theory of Transactional Memory (WTTM)*, Jerusalem, Israel.

[Riegel et al., 2011] Riegel, T., Marlier, P., Nowack, M., Felber, P., and Fetzer, C. (2011).
In *Proc. of the 23rd ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 53–64, San Jose, CA.