Lecture 28 - Software Transactional Memory

ECE 459: Programming for Performance

Jon Eyolfson

University of Waterloo

March 26, 2012

Introduction

- An old idea, that is seeing some renewed interest

- Instead of programming with locks, we have transactions with memory
    - Analogous to database transactions

- Either a series of memory operations all succeed or fail (and get rolled back) and are later retried

Benefit

- Very simple programming model, you don't have to worry about lock granularity or deadlocks
- You just group lines of code that should logically be one operation in an `atomic` block
- It is then the implementations job to make sure that if the code operates as if it is an atomic transaction

Example

```
transfer_funds(Account* sender, Account* receiver,
               double amount) {
  atomic {
    sender->funds -= amount;
    receiver->funds += amount;
  }
}
```

- With locks we have two main options:
  - Lock everything to do with modifying accounts (slow, may forget to use lock)
  - Have a lock for every account (deadlocking, may forget to use lock)
- With STM, we do not have to worry about remembering to acquire locks or deadlocks

Drawbacks

- The concept of rollback is key, however some things can not be rolled back (write to the screen, packet over the network)

- Nested transactions, what if an inner transaction succeeds, yet the transaction aborts?

- Most implementatoins (especially hardware) have a limited transaction size

Basic Implementation (Software)

- In all atomic blocks all reads/writes are recorded to a log
- At the end of the block, the thread verifies that no other threads have modified any values read
- If the validation is successful, the changes are **committed**
- Otherwise, the block is **aborted** and re-executed

Note: there are also hardware implementations as well

Basic Implementation Issues

- Since you don't protect against dataraces and just rollback it is possible for a datarace to trigger a fatal error in your program

```
atomic {
  x++;
  y++;
}
```

```
atomic {
  if (x != y)
     while (true) { }
}
```

- In this silly example, initially `x = y` and you may think the code will not go into an infinite loop, but it can

Implementations

Note: Typically the performance is no worse than twice as slow over fine-grained locks

- Toward.Boost.STM (C++)
- SXM (Microsoft, C#)
- Built-in to the language (Clojure, Haskell)
- AtomJava (Java)
- Durus (Python)

Summary

- Software Transactional Memory provides a more natural approach to parallel progrmaming
- No need to deal with locks and associated problems
- Currently slow, but a lot of research is going into improving this now
- Operates by either completing an atomic block or retrying (by rolling back) until it successfully completes