Transactional Memory

David Chisnall

March 10, 2011
What is a Transaction?

• Set of operations
• Operates atomically
• Either all succeed, or all fail
History: Databases

ACID compliance (simplified):

**Atomicity** - groups of commands must execute as a single operation

**Consistency** - the database must never expose partially completed operations

**Isolation** - concurrent operations should not have to worry about each other

**Durability** - the database shouldn’t randomly corrupt data
BEGIN TRANSACTION;
 SELECT {stuff};
 UPDATE {more stuff};
 UPDATE {even more stuff};
 END TRANSACTION;

- The statements in the transaction that modify the database either all complete or all fail.
- If they fail, you can detect this and retry.
Transactions for Concurrency

- Two users issue transactions
- Both see the data in a consistent state
- If both modify the same data, (at least) one fails
- Neither sees the data in the middle of the other’s changes
Transactional Memory

- Transactions are great for concurrency
- Why not make them available for all memory?
Simplest Case: Load-Linked / Store-Conditional

- Found on ARM, PowerPC, etc. chips
- Load-linked begins transaction on a word
- Store-conditional commits it
- Limitation: Transaction may only modify one word in memory
General Case: Fully Transactional Memory

- All modifications to memory are private
- Commit instruction applies them all
- Commit instruction fails if any updated memory locations has been modified
- Multiword Load-Linked / Store-Conditional
The Problem

- Must keep a copy of all memory updated by in-flight transactions
- Potentially huge!
- Must lock all memory locations for update
Hardware Transactional Memory

- Add instructions for begin / commit transaction
- Handle all updates via caches / buffers
- Not implemented anywhere (yet!)
Hardware-Assisted Transactional Memory: Rock

- Rock was an UltraSPARC design from Sun
- Project was cancelled near completion by Oracle
- Provided some HTM support
Rock’s Interfaces

- New `chkpt` and `commit` instructions
- `chkpt` looks like conditional branch
- Branches if `commit` fails
- Instructions between `chkpt` and `commit` not committed if conflict occurred
- Lots of limitations: No function calls, small number of memory addresses, no ‘difficult’ instructions in transactions...
- Can be used to assist implementing software transactional memory
Software Transactional Memory

- Emulate hardware transactional memory
- Comes with performance overhead
- Can still be useful
STM Approaches: Indirection and Swizzling

- Create a copy of data to modify
- Modify copy
- Update pointer with atomic compare-and-swap
- Transaction fails if pointer has been modified since copy was made
- Uses a lot of memory (and time to make all of the copies)
- Simple to implement, works well for small data structures with a single entry point
STM Approaches: Locking

- Lock(s) must be held for the duration of an update
- Global lock? Very little concurrency possible
- Locks on ranges of memory (e.g. one per page)
- Deadlock potential: must acquire locks in same order (e.g. lowest memory address first)
Why Transactional Memory is *Damn Shiny™*

- With atomic set and get, you can’t make atomic increment
- With transactional set and get, you can
- **Arbitrary transactions are composable**
void atomic_increment(void)
{
    do
    {
        begin_transaction();
        int a = atomic_get();
        a++;
        atomic_set(a);
    } while (end_transaction());
}
Proposed Imperative Language Support

```c
atomic
{
    if (i == 0)
        retry;
    next = queue[i];
    i++;
};
```

- **atomic** keyword defines a transaction
- **retry** restarts transaction, blocks until a memory address that has been read already is modified by something else
- Transaction automatically restarted if it fails
Example: Transactional Linked List

```c
void insert(struct list_item *li);
{
    atomic
    {
        li->prev = NULL;
        li->next = head->next;
        head = li;
    }
}
```
STM and Haskell

- Fitting transactional memory into imperative programming is hard
- Haskell’s type system already collects side effects in monads
- Concurrency monad implements STM
- Monad handles merging results, retrying failed transactions
Questions?