Transactional Memory

Active research! Here be dragons…

BRACE YOURSELVES

COMPLEX BUT COOL IDEAS ARE COMING
Challenges of Synchronization

- Two major issues:
  - Performance
    - Scalability
  - Base cost
    - We have looked at some techniques that address this
      - Better spinlocks
      - Lockless strategies (NBS, RCU)
  - Programmability
    - Locks are hard to use correctly
    - Lockless data structures are hard to design
What’s missing?

• Lack of support for *abstraction* and *composition*

• E.g. Suppose we have thread-safe stack with (abstract) push and pop operations
  • In sequential programs, can use these operations without regard to their implementation
  • In parallel programs, internal details may be needed
    • Consider task of moving an item from one stack to another
    • Need to expose stack locking mechanism
“Magic” Wish List

• Let programmers express desired outcome
  • “This block of code should appear atomic”
• Let run-time system or hardware support make it happen
• Allow abstractions to hide implementation and be composable

拇指向下 A new programming model is needed
Database Transactions

- Database systems allow multiple queries to run in parallel
- Query authors don’t worry about concurrency
- Complex queries can be composed out of simpler ones
- Can we use the DB programming model as a general parallel model?
- **Key Programming Model:** everything is a transaction
  - A transaction executes as if it were the only computation accessing the database
  - Restricted interactions, serializability
  - Hide complex implementation detail, programmer only sees a simple interface
  - Atomic – all updates become visible, or none
  - Consistent – transactions leave database in consistent state
  - Isolated – no interference with or from other transactions
  - Durable – once committed, updates are permanent
Transactional Memory: Some History

- 1977 – D.B. Lomet (IBM Research, now at Microsoft Research) suggests database transaction model for concurrent programming
  - No practical implementation provided
- 1983 – Kung & Robinson propose optimistic concurrency control for databases
- 1988 – Chang & Mergen describe IBM 801 storage manager
  - HW provided lock bits for each 128 byte range of a page; page tables & TLB extended
- 1993 – Herlihy & Moss describe a hardware proposal for transactional memory
Transactional Memory (TM)

Source Code:

```c
atomic {
    ...
    access_shared_data();
    ...
}
...```

Transactions:

TM System:

Programmer: Specifies threads/transactions in source code
TM System: Executes transactions optimistically in parallel
1) Checkpoints execution
2) Detects conflicts
3) Commits or aborts and re-executes
Differences from DB Transactions

• Memory vs. disk
  • Disk access takes 100X longer than memory access => database systems can use relatively heavy-weight software solutions

• No need for durability
  • Memory is transient anyway => simplifies TM implementations

• Existing languages, libraries and systems
  • Databases are closed systems in which all code executes as a transaction, programs using TM must coexist with libraries, OSs that do not
TM Implementations

- Hardware TM (HTM)
  - Changes to computer system and ISA
  - Extra cache to buffer writes, extended coherence protocol to track conflicts, special transaction instructions
  - Support for limited number of memory locations
- Software TM (STM)
  - Language runtime (or library) + extensions to specify transaction
  - Exploit current commodity hardware (multicores)
  - Get experience with transactional programming model
  - Java: DSTM (Marathe et al.), ASTM (Herlihy et al.)
  - C/C++: McRT-STM (Saha et al.), TL2 (Dice et al.), RSTM
  - Intel’s C++ STM compiler
- Hybrid TM (HyTM)
Programming Constructs

- Atomic block

```java
atomic {
    if (x!=null) x.foo();
    y = true;
}
```

- Delimits code that should execute in a transaction
- Dynamically-scoped – code in foo() executes in transaction as well
- Does not name shared resources (unlike monitors or lock-based programming)
- 3 possible outcomes – commits, aborts, non-termination
Caution!

- Programmers can still use *atomic* incorrectly

```c
bool flagA=false; bool flagB=false;

Thread 1:
atomic {
    while (!flagA);
    flagB = true;
}

Thread 2:
atomic {
    flagA = true;
    while (!flagB);
}
```

- What’s wrong?
  - Deadlock results
Semantics

• Not yet formally specified!

• Useful ways to reason about TM:
  • Database correctness criteria: serializability
    • Useful for understanding transaction behaviour
    • Says nothing about interaction of transactions with code outside of transactions
  • Operational semantics – single-lock atomicity (SLA)
    • Program executes as if all atomic blocks were protected by single global lock
    • Attractive, but may be problematic conceptually
    • SLA does not support failure atomicity, forms of nesting, etc.
Implementation Basics

• For all (non-stack) write instructions:
  • Track write addresses and values (write set)
• For all (non-stack) read instructions:
  • track read addresses and values (read set)
• When a transaction completes:
  • Atomically
    • Validate read set (conflict detection)
  • Commit write set
Implementation Options

- **Transaction Granularity**
  - Unit of storage over which TM system detects conflicts
  - Similar to notion of cache coherence
  - Word or block typical for HTM, object common for STMs that extend OO language

- **Direct or Deferred Update**
  - Direct – transaction directly modifies the object itself
    - Must log previous value for undo in case of abort
  - Deferred – modify private copy, propagate at commit
  - Both get complicated in the presence of data races

- **Optimistic or Pessimistic Concurrency Control**
  - TM typically optimistic; need to detect and resolve conflict
Location-Based Conflict Detection

Transaction 1:
Strip versions:

Main Memory:
Strip versions:

Transaction 2:
Strip versions:

Legend:
- Read
- Written

Strips
Location-Based Conflict Detection

Transaction 1:  
Strip versions:

Main Memory:  
Strip versions:

Transaction 2:  
Strip versions:

Legend:
- Read
- Written
Location-Based Conflict Detection

Transaction 1:
Transaction 2:
Main Memory:

Strip versions:

Commit step 1) Validate Read Set ✅
Commit step 2) Publish Writes (and inc version #s)

Legend:
- Read
- Written
Location-Based Conflict Detection

**Transaction 1:**

Strip versions:

<table>
<thead>
<tr>
<th>Strip version</th>
<th>0</th>
<th>2</th>
<th>3</th>
<th>5</th>
</tr>
</thead>
</table>

Main Memory:

Strip versions:

<table>
<thead>
<tr>
<th>Location</th>
<th>0</th>
<th>6</th>
<th>9</th>
<th>3</th>
<th>5</th>
<th>0</th>
</tr>
</thead>
</table>

**Transaction 2:**

Strip versions:

<table>
<thead>
<tr>
<th>Strip version</th>
<th>0</th>
</tr>
</thead>
</table>

Commit step 1) Validate Read Set

Abort!

Note: all transactions must maintain strip version #s

Legend:

- Read
- Written
Value-Based Conflict Detection

Transaction 1:

Main Memory:

Transaction 2:

Legend:
- Read
- Written
Value-Based Conflict Detection

Transaction 1: 2 3 5

Main Memory: 6 2 3 5

Transaction 2: 6 9

Legend:
- Read
- Written

University of Toronto, Department of Computer Science
Value-Based Conflict Detection

Transaction 1:

Main Memory:

Transaction 2:

Commit step 1) Validate Read Set
Commit step 2) Publish Writes

Legend:

Read

Written
Value-Based Conflict Detection

Transaction 1:

Main Memory:

Transaction 2:

Commit step 1) Validate Read Set ✗ Abort!

Note: no version information to maintain

Legend:

Read

Written
TM Weaknesses

• Some operations are hard to abort/retry
  • Essentially anything not idempotent, e.g. I/O
• In practice, TM does not interact well with locking
• Some variables are prone to high conflict rates (frequent true sharing & dependences)
• Conflict resolution needs to avoid starving long-running, large transactions
• Poor interaction with standard software tools like debuggers
  • Getting better though ...
TM Status

- Hardware TM is now a reality
  - Sun’s Rock processor was killed after acquisition by Oracle (2009)
  - Azul Systems has HTM in their Java appliance hardware (circa 2009)
  - IBM BlueGene/Q (2011)
  - Intel Haswell’s *Transactional Sync Extensions (TSX)*
- Software TM has performance problems
  - But some applications are a nice fit
    - E.g. parallel game server
Announcements

• Tutorial will cover A2 discussion and clarifications
• Make sure to attend!
• Ask questions, prepare in advance by reading what you have to do
• Read the Hoard paper, make sure you understand the details
• Reminder: start early!
• Keep in mind: for A3 you can only use up to 1 grace token! (See info sheet)
Midterm marks (avg: 69%)

- Make sure to pick up your midterm during office hours
- Check out solutions and review your midterm
- If you have questions, please ask me!