Why The Grass May Not Be Greener On The Other Side:
A Comparison of Locking vs. Transactional Memory

Paul E. McKenney, IBM Linux Technology Center
Maged M. Michael, IBM TJ Watson Research
Jonathan Walpole, Portland State University
Overview, Rationale, and Methodology

- Inexpensive multi-threaded/multi-core CPUs are here!
- Typical practitioner now must handle concurrency
- Transactional memory seen as one possible solution
  - But need to compare fairly to existing mechanism: locking
  - Comparison must cover all relevant attributes
  - But balanced comparisons are difficult in “hot” fields like TM
- Methodology for balanced comparison:
  - Maged Michael: strong NBS background, working with STM
  - Paul McKenney: strong locking/RCU background
  - Jon Walpole: versatile, strong conflict-resolution skills
- Any characterization of locking and TM that Maged and Paul can agree is necessarily very well balanced
## Locking and TM: Basics

<table>
<thead>
<tr>
<th></th>
<th>Locking</th>
<th>Transactional Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Basic Idea</strong></td>
<td>Allow only one thread at a time to access a given set of objects.</td>
<td>Cause a given operation over a set of objects to execute atomically.</td>
</tr>
<tr>
<td><strong>Scope</strong></td>
<td>Idempotent and non-idempotent operations.</td>
<td>Idempotent operations. Non-idempotent operations require hacks.</td>
</tr>
<tr>
<td><strong>Composability</strong></td>
<td>Limited by deadlock.</td>
<td>Limited by non-idempotent operations and performance.</td>
</tr>
<tr>
<td><strong>Scalability and Performance</strong></td>
<td>Data must be partitionable to avoid lock contention.</td>
<td>Data must be partitionable to avoid conflicts.</td>
</tr>
<tr>
<td></td>
<td>Partitioning typically must be fixed at design time.</td>
<td>Dynamic adjustment of partitioning carried out automatically.</td>
</tr>
<tr>
<td></td>
<td>Contention effects can be focused on acquisition and release, so that critical section runs at full speed.</td>
<td>Contention effects can degrade performance of processing within the transaction.</td>
</tr>
<tr>
<td></td>
<td>FIFO locking primitives can provide deterministic response for bounded number of threads.</td>
<td>Deterministic response may require sacrificing other requirements.</td>
</tr>
</tbody>
</table>
Locking and TM: Practical Applicability

<table>
<thead>
<tr>
<th></th>
<th>Locking</th>
<th>Transactional Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>HW Support</td>
<td>Commodity hardware suffices.</td>
<td>New hardware required, else performance limited by STM.</td>
</tr>
<tr>
<td></td>
<td>Performance insensitive to details of cache geometry.</td>
<td>HTM performance depends critically on cache geometry.</td>
</tr>
<tr>
<td>SW Support</td>
<td>APIs exist, large body of code and experience, debuggers operate naturally.</td>
<td>APIs emerging, little experience outside of DBMS, breakpoints mid-transaction can be problematic.</td>
</tr>
<tr>
<td>Practical applications exist</td>
<td>Yes.</td>
<td>Yes.</td>
</tr>
<tr>
<td>Wide applicability</td>
<td>Yes.</td>
<td>Jury still out.</td>
</tr>
</tbody>
</table>
Status of STM and HTM

- There are cases where STM works very well
  - Scalability can overcome overhead penalty
  - In some special cases, with as few as 4 CPUs

- In other cases, STM is more painful
  - 20x or, in rare cases, 100x overhead vs. uncontended locking

- There are some indications that HTM falling back to STM incurs significantly greater overhead than pure STM
  - Hardware acceleration for STM?

- STM can be tailored for specific applications
Where Do Locking and TM Fit In?

**Locking:**
- Non-idempotent operations
- Large critical sections
- High performance on commodity hardware
- Good scalability given good engineering (Linux on 1024 CPUs)
  - When data is statically partitionable
- Large body of successful practice and experience
- Excellent performance and scalability on read-mostly data
  - In conjunction with RCU or hazard pointers

**TM:**
- Large partitionable data structures that lack static partitionability
- Situations where no clear lock hierarchy exists (avoid deadlock)
- Single-threaded software with embarrassingly parallel core
- TM's applicability may increase if STM performance improves
Conclusion: Use the Right Tool For The Job!!!

- There is no silver bullet: successful adoption of multi-threaded/multi-core CPUs will require combination of techniques
  - But don't take our word for it, ask the TxLinux guys 😊
- Analogy with engineering: How many types of fasteners are there? How many subtypes? Nail, screw, clip, bolt, glue, joint, magnet...
- Neither locking nor TM solve the fundamental performance and scalability problems (later slides cover ease of use)
  - STM struggling to achieve parity with uncontended locking, HTM performance benefits over uncontended locking appear to be quite limited
    - Which is a source of much amusement to those of us who have designed and implemented deadlock-immune mechanisms more than an order of magnitude faster than uncontended locking (RCU and Hazard Pointers)
- Future work: Relativistic Programming
  - Formalize and generalize existing techniques such as RCU
  - Integrate with other techniques: “use the right tool for the job”
  - Combine performance, scalability, and ease of use
  - Account for common hardware properties
    - Allow hardware designers freedom to improve performance
Corroboration From SOSP 2007 TxLinux Paper

- **Tried transactions**: 6-year effort, difficult change
- **Used locking/transaction hybrid approach**: 1 month
  - Modest performance gains of ~2%
    - Even with favorable-to-TxLinux single-cycle-per-instruction assumption
    - Contrast with tens-of-percent and order-of-magnitude increases from other changes
  - Locking required for I/O and runqueue locks
  - Because TxLinux falls back to locking, deadlock can still arise
    - “While this is unfortunate, deadlock is also a possibility for advanced transaction models that allow open nesting.”
    - Suggested solution: use single global lock for transactions that are unlikely to fail
    - However, additional deadlock scenarios are generated by hybrid approach!!!
    - Question: has TxLinux really delivered on the ease-of-programming TM promise?
  - Encountered priority inversion, requiring scheduler support

- **In short, TM is not immune to vicissitudes of large and complex real-world software artifacts**
  - Question: suppose TxLinux team had instead applied TM to a few key areas in the Linux kernel where deadlock avoidance results in complex code?
    - Might doing so result in a large removed-lines-of-code metric?
Legal Statement

- This work represents the view of the author and does not necessarily represent the view of IBM.

- IBM, IBM (logo), e-business (logo), pSeries, e (logo) server, and xSeries are trademarks or registered trademarks of International Business Machines Corporation in the United States and/or other countries.

- Linux is a registered trademark of Linus Torvalds.

- Other company, product, and service names may be trademarks or service marks of others.
Discussion