Is Transactional Programming Actually Easier?

Christopher J. Rossbach, Owen S. Hofmann, Emmett Witchel
University of Texas at Austin, USA
Transactional Memory: Motivation Mantra

- We need better parallel programming tools
  - (Concurrent programming == programming w/locks)
  - Locks are difficult
  - CMP ubiquity $\rightarrow$ urgency
- Transactional memory is “promising”:
  - No deadlock, livelock, etc.
  - Optimistic $\rightarrow$ likely more scalable
- Conclusion:
  - Transactional Memory is *easier* than locks
  - Corollary: All TM papers should be published
Is TM *really* easier than locks?

- Programmers *still must write critical sections*
- Realizable TM will have *new issues*
  - HTM overflow
  - STM performance
  - Trading one set of difficult issues for another?
- Ease-of-use is a critical motivator for TM research

*It’s important to know the answer to this question*
How can we answer this question?

Step 1: Get some programmers (preferably inexperienced)

Step 2: have them write the same program with TM and locks

Step 3: Ask them how it went

Step 4: Evaluate their code

This talk:
• TM vs. locks user study
• UT Austin OS undergrads
• same program using
  • locks (fine/coarse)
  • monitors
  • transactional memory
Outline

- Motivation
- Programming Problem
- User Study Methodology
- Results
- Conclusion
The programming problem

sync-gallery: a rogue’s gallery of synchronization

- Metaphor → shooting gallery (welcome to Texas)
- Rogues → shoot paint-balls in lanes (1 red, 1 blue)
- Cleaners → change targets back to white
Sync-gallery invariants

- Only one shooter per lane (Uh, hello, dangerous?!)  
- Don’t shoot colored lanes (no fun)  
- Clean only when all lanes shot (be lazy)  
- Only one cleaner at a time
Sync-gallery Implementations

- Program specification variations
  - Single-lane
  - Two-lane
  - Cleaner (condition vars + additional thread)

- Synchronization primitive variations
  - Coarse: single global lock
  - Fine: per lane locks
  - Transactional Memory
Variation 1: "single-lane rogue"

Rogue() {
    while(true) {
        Lane lane = randomLane();
        if(lane.getColor() == WHITE)
            lane.shoot();
        if(allLanesShot())
            clean();
    }
}

Invariants:
• One shooter per lane
• Don’t shoot colored lanes
• One cleaner thread
• Clean only when all lanes shot
Variation 2: “two-lane rogue”

Rogue() {
    while(true) {
        Lane a = randomLane();
        Lane b = randomLane();
        if(a.getColor() == WHITE &&
           b.getColor() == WHITE) {
            a.shoot();
            b.shoot();
        }
        if(allLanesShot())
            clean();
    }
}

Global Locking

Invariants:
• One shooter per lane
• Don’t shoot colored lanes
• One cleaner thread
• Clean only when all lanes shot

EmbraceINDependence
Variation 3: “cleaner rogues”

Rogue() {
    while(true)
        Lane lane = randomLane();
        if(lane.getColor() == WHITE) if(allLanesShot()) lane.shoot();
    }

Cleaner() {
    while(true) {
        if(allLanesShot()) clean();
    }
}

{still need other locks!}

Invariants:
• One shooter per lane
• Don’t shoot colored lanes
• One cleaner thread
• Clean only when all lanes shot
Synchronization Cross-product

<table>
<thead>
<tr>
<th></th>
<th>Coarse</th>
<th>Fine</th>
<th>TM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single-lane</td>
<td>Coarse</td>
<td>Fine</td>
<td>TM</td>
</tr>
<tr>
<td>Two-lane</td>
<td>Coarse2</td>
<td>Fine2</td>
<td>TM2</td>
</tr>
<tr>
<td>Cleaner</td>
<td>CoarseCleaner</td>
<td>FineCleaner</td>
<td>TMCleaner</td>
</tr>
</tbody>
</table>

9 different Rogue implementations
Outline

- Motivation
- Programming Problem
- User Study Methodology
  - TM Support
  - Survey details
- Results
- Conclusion
TM Support

- Year 1: DSTM2 [Herlihy 06]
- Year 2+3: JDASTM [Ramadan 09]
- Library, not language support
  - No atomic blocks
  - Read/write barriers encapsulated in lib calls
  - Different concrete syntax matters
Callable c = new Callable<Void> {
    public Void call() {
        GalleryLane l = randomLane();
        if(l.color() == WHITE)
            l.shoot(myColor);
        return null;
    }
};
Thread.doIt(c); // ← transaction here
JDASTM concrete syntax

```
Transaction tx = new Transaction(id);
boolean done = false;
while(!done) {
    try {
        tx.BeginTransaction();
        GalleryLane l = randomLane();
        if(l.TM_color() == WHITE))
            l.TM_shoot(myColor);
        done = tx.CommitTransaction();
    } catch(AbortException e) {
        tx.AbortTransaction();
        done = false;
    }
}
```
Undergrads: the ideal TM user-base

- TM added to undergrad OS curriculum
- Survey accompanies sync-gallery project
- Analyze programming mistakes
- TM’s benchmark for success
  - *Easier to use than fine grain locks or conditions*
Survey

- Measure previous exposure
  - Used locks/TM before, etc
- Track design/code/debug time
- Rank primitives according along several axes:
  - Ease of reasoning about
  - Ease of coding/debugging
  - Ease of understanding others’ code

Data collection

- Surveyed 5 sections of OS students
  - 2 sections x 2 semesters + 1 section x 1 semester
  - 237 students
  - 1323 rogue implementations
- Defect Analysis
  - Automated testing using condor
  - Examined all implementations
Outline

- Motivation
- Programming Problem
- User Study Methodology
- Results
- Conclusion
Development Effort: year 2

<table>
<thead>
<tr>
<th></th>
<th>coarse</th>
<th>fine</th>
<th>tm</th>
<th>coarse</th>
<th>fine</th>
<th>tm</th>
<th>coarse</th>
<th>fine</th>
<th>tm</th>
<th>coarse</th>
<th>fine</th>
<th>tm</th>
</tr>
</thead>
<tbody>
<tr>
<td>single-lane</td>
<td></td>
<td></td>
<td></td>
<td>two-lane</td>
<td></td>
<td></td>
<td>cleaner</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **hours**
  - 0
  - 0.5
  - 1
  - 1.5
  - 2
  - 2.5
  - 3
  - 3.5
  - 4

- **categories**
  - debug (green)
  - code (blue)
  - design (red)
Development Effort: year 2

Implementation order:

Coarse

rand&2?

Fine

TM

design
code
dbg
Development Effort: year 2

- Single-lane: coarse, fine, tm
- Two-lane: coarse, fine, tm
- Cleaner: coarse, fine, tm

- Hours:
  - 0
  - 0.5
  - 1
  - 1.5
  - 2
  - 2.5
  - 3
  - 3.5
  - 4

- Categories:
  - Debug
  - Code
  - Design
Development Effort: year 2

- **Coarse**
  - Single-lane: 3
  - Two-lane: 2
  - Cleaner: 1

- **Fine**
  - Single-lane: 2
  - Two-lane: 1
  - Cleaner: 1

- **TM**
  - Single-lane: 1
  - Two-lane: 1
  - Cleaner: 1

Legend:
- **Debug**
- **Code**
- **Design**
## Qualitative preferences: year 2

### Best Syntax

<table>
<thead>
<tr>
<th>Ranking</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coarse</td>
<td>62%</td>
<td>30%</td>
<td>1%</td>
<td>4%</td>
</tr>
<tr>
<td>Fine</td>
<td>6%</td>
<td>21%</td>
<td>45%</td>
<td>40%</td>
</tr>
<tr>
<td>TM</td>
<td>26%</td>
<td>32%</td>
<td>19%</td>
<td>21%</td>
</tr>
<tr>
<td>Conditions</td>
<td>6%</td>
<td>21%</td>
<td>29%</td>
<td>40%</td>
</tr>
</tbody>
</table>

### Easiest to Think about

<table>
<thead>
<tr>
<th>Ranking</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coarse</td>
<td>81%</td>
<td>14%</td>
<td>1%</td>
<td>3%</td>
</tr>
<tr>
<td>Fine</td>
<td>1%</td>
<td>38%</td>
<td>30%</td>
<td>29%</td>
</tr>
<tr>
<td>TM</td>
<td>16%</td>
<td>32%</td>
<td>30%</td>
<td>21%</td>
</tr>
<tr>
<td>Conditions</td>
<td>4%</td>
<td>14%</td>
<td>40%</td>
<td>40%</td>
</tr>
</tbody>
</table>
### Qualitative preferences: year 2

<table>
<thead>
<tr>
<th>Easiest to Think about</th>
<th>Ranking</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coarse</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>62%</td>
<td>30%</td>
<td>1%</td>
<td>4%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6%</td>
<td>26%</td>
<td>32%</td>
<td>45%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>26%</td>
<td>32%</td>
<td>19%</td>
<td>21%</td>
</tr>
<tr>
<td>Conditions</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>6%</td>
<td>21%</td>
<td>29%</td>
<td>40%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6%</td>
<td>21%</td>
<td>29%</td>
<td>40%</td>
</tr>
</tbody>
</table>

| Conditions            |         |       |       |       |       |
|                        |         | 6%    | 21%   | 29%   | 40%   |

#### Best Syntax
- Easiest to Think about
  - Coarse
  - Fine
  - TM
- Conditions
  - Easier
  - Harder

(Year 2)
### Qualitative preferences: year 2

#### Best Syntax

<table>
<thead>
<tr>
<th>Ranking</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coarse</td>
<td>62%</td>
<td>30%</td>
<td>1%</td>
<td>4%</td>
</tr>
<tr>
<td>Fine</td>
<td>6%</td>
<td>21%</td>
<td>45%</td>
<td>40%</td>
</tr>
<tr>
<td>TM</td>
<td>26%</td>
<td>32%</td>
<td>19%</td>
<td>21%</td>
</tr>
<tr>
<td>Conditions</td>
<td>6%</td>
<td>21%</td>
<td>19%</td>
<td>40%</td>
</tr>
</tbody>
</table>

#### Easiest to think about

<table>
<thead>
<tr>
<th>Ranking</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coarse</td>
<td>81%</td>
<td>14%</td>
<td>1%</td>
<td>3%</td>
</tr>
<tr>
<td>Fine</td>
<td>1%</td>
<td>38%</td>
<td>30%</td>
<td>29%</td>
</tr>
<tr>
<td>TM</td>
<td>16%</td>
<td>32%</td>
<td>30%</td>
<td>21%</td>
</tr>
<tr>
<td>Conditions</td>
<td>4%</td>
<td>14%</td>
<td>40%</td>
<td>40%</td>
</tr>
</tbody>
</table>
Analyzing Programming Errors

Error taxonomy: 10 classes

- **Lock-ord**: lock ordering
- **Lock-cond**: checking condition outside critsec
- **Lock-forgot**: forgotten Synchronization
- **Lock-exotic**: inscrutable lock usage
- **Cv-exotic**: exotic condition variable usage
- **Cv-use**: condition variable errors
- **TM-exotic**: TM primitive misuse
- **TM-forgot**: Forgotten TM synchronization
- **TM-cond**: Checking conditions outside critsec
- **TM-order**: Ordering in TM
Error Rates by Defect Type

- cv-use: 63%
- TM: 0-27%

Bar chart showing error rates for different defect types.
Overall Error Rates

Geometric Mean:
- Coarse Locks: 27%
- Fine Locks: 62%
- TM: 10%

Locks: 58-75%
TM: 8-20%
Outline

- Motivation
- Programming Problem
- User Study Methodology
- Results
- Conclusion
Conclusion

- General qualitative ranking:
  1. Coarse-grain locks (easiest)
  2. TM
  3. Fine-grain locks/conditions (hardest)
- Error rates overwhelmingly in favor of TM
- TM may *actually be easier*
Overall Error Rates: Year 2

Proportion of errors

Coarse, Fine, TM, Coarse2, Fine2, TM2, CoarseCleaner, FineCleaner, TMCleaner