Is Transactional Programming Actually Easier?

Christopher J. Rossbach, Owen S. Hofmann, Emmett Witchel
UT Austin
TM Research Mantra

- We need better parallel programming tools
  - CMP ubiquity
  - (Concurrent programming == programming w/locks)
  - Locks are difficult
- Transactional memory is “promising”:
  - No deadlock, livelock, etc.
  - Optimistic → likely more scalable
- Therefore:
  - Transactional Memory is *easier* than locks
  - 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 using 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 UT)
- Rogues → shoot paint-balls in lanes
  - Each rogue has a unique color
- Shooting → target takes rogue’s color
- Cleaners → change targets back to white
- Rogues/cleaners must synchronize
  - maintain 4 invariants
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 thread
Task: “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: “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();
    }
}

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

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

Cleaner() {
    while(true) {
        if(allLanesShot())
            lanesFull.signal();
        clean();
    }
}

(still need other locks!)

Invariants:
• One shooter per lane
• Don’t shoot colored lanes
• One cleaner thread
• Clean only when all lanes shot
Sync-gallery in action
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: JDASTM [Ramadan 09]
- Library, not language support
  - No atomic blocks
  - Different concrete syntax
  - Read/write barriers
Callable c = new Callable<Void> {
    public Void call() {
        GalleryLane l = randomLane();
        if(l.color() == WHITE))
            l.shoot(myColor);
        return null;
    }
}
Thread.doIt(c);
Transaction tx = new Transaction(id);
boolean done = false;
while(!done) {
    try {
        tx.BeginTransaction();
        GalleryLane l = randomLane();
        if(l.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 students
- 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 4 sections of OS students
  - 2 sections x 2 semesters
  - 147 students
  - 1323 rogue implementations
- Defect Analysis
  - Examined year 2 (604 implementations)
  - Automated testing using condor
Outline

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

- hours
- coarse
- fine
- tm
- single-lane
- two-lane
- cleaner
- debug
- code
- design
Development Effort

![Graph showing development effort with hours on the y-axis and coarse, fine, and tm on the x-axis. The graph compares single-lane, two-lane, and cleaner with different effort levels.]

Implementation order:

- Coarse
- rand&2?
- Fine
- TM

Order of implementation:

1. Coarse
2. rand&2?
3. Fine
4. TM
<table>
<thead>
<tr>
<th></th>
<th>coarse</th>
<th>fine</th>
<th>tm</th>
</tr>
</thead>
<tbody>
<tr>
<td>single-lane</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>two-lane</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cleaner</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Development Effort**

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

- **Categories**
  - debug
  - code
  - design

The chart shows the development effort for coarse, fine, and tm (target movement) in single-lane, two-lane, and cleaner scenarios.
## Qualitative preferences

### 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>

(Year 2)
## Qualitative preferences

### Best Syntax

| Ranking | Coarse | (TM|Coarse) | Fine | (Fine|Conditions) |
|---------|--------|------------|------|-----------------|
| 1       | 81%    | 14%        | 1%   | 3%              |
| 2       | 62%    | 30%        | 1%   | 4%              |
| 3       | 32%    | 21%        | 40%  | 4%              |
| 4       | 19%    | 14%        | 30%  | 21%             |

### Easiest to Think about

| Ranking | Coarse | (Fine|TM) | Conditions |
|---------|--------|------|------------|
| 1       | 38%    | 30%  | 29%        |
| 2       | 32%    | 30%  | 21%        |
| 3       | 32%    | 40%  | 40%        |
| 4       | 14%    | 14%  | 40%        |

(Year 2)
Analyzing Programming Errors

Error taxonomy: 8 classes

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

- Lock-ord: 8%
- Lock-cond: 7%
- Lock-forgot: 15%
- Cv-exotic: 8%
- Cv-use: 11%
- TM-exotic: 3%
- TM-order: 2%
- TM-forgot: 0.50%
Overall Error Rates

![Overall Error Rates Graph](image-url)
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...
## P-values

<table>
<thead>
<tr>
<th></th>
<th>Coarse</th>
<th>Fine</th>
<th>TM</th>
<th>Coarse2</th>
<th>Fine2</th>
<th>TM2</th>
<th>CoarseCleaner</th>
<th>FineCleaner</th>
<th>TMCleaner</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coarse</td>
<td>Y1</td>
<td>1.00</td>
<td>0.03</td>
<td>0.02</td>
<td>1.00</td>
<td>0.02</td>
<td>1.00</td>
<td>0.95</td>
<td>0.47</td>
</tr>
<tr>
<td></td>
<td>Y2</td>
<td>1.00</td>
<td>0.33</td>
<td>0.12</td>
<td>1.00</td>
<td>0.38</td>
<td>1.00</td>
<td>1.00</td>
<td>0.18</td>
</tr>
<tr>
<td>Fine</td>
<td>Y1</td>
<td>0.97</td>
<td>1.00</td>
<td>0.33</td>
<td>1.00</td>
<td>0.24</td>
<td>1.00</td>
<td>1.00</td>
<td>0.97</td>
</tr>
<tr>
<td></td>
<td>Y2</td>
<td>0.68</td>
<td>1.00</td>
<td>0.58</td>
<td>1.00</td>
<td>0.51</td>
<td>1.00</td>
<td>1.00</td>
<td>0.40</td>
</tr>
<tr>
<td>TM</td>
<td>Y1</td>
<td>0.98</td>
<td>0.68</td>
<td>1.00</td>
<td>1.00</td>
<td>0.13</td>
<td>1.00</td>
<td>1.00</td>
<td>0.98</td>
</tr>
<tr>
<td></td>
<td>Y2</td>
<td>0.88</td>
<td>0.43</td>
<td>1.00</td>
<td>1.00</td>
<td>0.68</td>
<td>1.00</td>
<td>1.00</td>
<td>0.41</td>
</tr>
<tr>
<td>Coarse2</td>
<td>Y1</td>
<td>&lt;0.01</td>
<td>&lt;0.01</td>
<td>&lt;0.01</td>
<td>1.00</td>
<td>&lt;0.01</td>
<td>&lt;0.01</td>
<td>&lt;0.01</td>
<td>&lt;0.01</td>
</tr>
<tr>
<td></td>
<td>Y2</td>
<td>&lt;0.01</td>
<td>&lt;0.01</td>
<td>&lt;0.01</td>
<td>1.00</td>
<td>&lt;0.01</td>
<td>0.45</td>
<td>&lt;0.01</td>
<td>&lt;0.01</td>
</tr>
<tr>
<td>Fine2</td>
<td>Y1</td>
<td>0.98</td>
<td>0.77</td>
<td>0.87</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td></td>
<td>Y2</td>
<td>0.62</td>
<td>0.49</td>
<td>0.32</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>0.99</td>
<td>0.95</td>
</tr>
<tr>
<td>TM2</td>
<td>Y1</td>
<td>&lt;0.01</td>
<td>&lt;0.01</td>
<td>&lt;0.01</td>
<td>0.99</td>
<td>&lt;0.01</td>
<td>1.00</td>
<td>0.04</td>
<td>&lt;0.01</td>
</tr>
<tr>
<td></td>
<td>Y2</td>
<td>&lt;0.01</td>
<td>&lt;0.01</td>
<td>&lt;0.01</td>
<td>0.55</td>
<td>&lt;0.01</td>
<td>1.00</td>
<td>&lt;0.01</td>
<td>&lt;0.01</td>
</tr>
<tr>
<td>CoarseCleaner</td>
<td>Y1</td>
<td>0.05</td>
<td>&lt;0.01</td>
<td>&lt;0.01</td>
<td>1.00</td>
<td>&lt;0.01</td>
<td>0.96</td>
<td>1.00</td>
<td>&lt;0.01</td>
</tr>
<tr>
<td></td>
<td>Y2</td>
<td>&lt;0.01</td>
<td>&lt;0.01</td>
<td>&lt;0.01</td>
<td>0.55</td>
<td>&lt;0.01</td>
<td>1.00</td>
<td>1.00</td>
<td>&lt;0.01</td>
</tr>
<tr>
<td>FineCleaner</td>
<td>Y1</td>
<td>0.53</td>
<td>0.03</td>
<td>0.02</td>
<td>1.00</td>
<td>&lt;0.01</td>
<td>1.00</td>
<td>0.99</td>
<td>1.00</td>
</tr>
<tr>
<td></td>
<td>Y2</td>
<td>0.83</td>
<td>0.60</td>
<td>0.59</td>
<td>1.00</td>
<td>0.42</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>TMCleaner</td>
<td>Y1</td>
<td>0.28</td>
<td>0.12</td>
<td>0.08</td>
<td>1.00</td>
<td>0.03</td>
<td>1.00</td>
<td>0.92</td>
<td>0.55</td>
</tr>
<tr>
<td></td>
<td>Y2</td>
<td>&lt;0.01</td>
<td>&lt;0.01</td>
<td>&lt;0.01</td>
<td>0.99</td>
<td>&lt;0.01</td>
<td>1.00</td>
<td>&lt;0.01</td>
<td>1.00</td>
</tr>
<tr>
<td>Rogue</td>
<td>Synchronization</td>
<td>R/C Threads</td>
<td>Additional Requirements</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>--------------</td>
<td>--------------------------------</td>
<td>-------------</td>
<td>-------------------------</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Coarse</td>
<td>Global lock</td>
<td>not distinct</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Coarse2</td>
<td>Global lock</td>
<td>not distinct</td>
<td>Shoot at 2 lanes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CoarseCleaner</td>
<td>Global lock, conditions</td>
<td>Distinct</td>
<td>wait/notify</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fine</td>
<td>Per-lane locks</td>
<td>not distinct</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fine2</td>
<td>Per-lane locks</td>
<td>not distinct</td>
<td>Shoot at 2 lanes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FineCleaner</td>
<td>Per-lane locks, conditions</td>
<td>Distinct</td>
<td>wait/notify</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TM</td>
<td>TM</td>
<td>not distinct</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TM2</td>
<td>TM</td>
<td>not distinct</td>
<td>Shoot at 2 lanes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TMCleaner</td>
<td>TM</td>
<td>distinct</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>