Modeling Transactional Memory
Workload Performance

Donald E. Porter and Emmett Witchel
The University of Texas at Austin
{porterde, witchel}@cs.utexas.edu

Transaction Performance

Transactions simplify writing correct programs …but they complicate performance.
• Transactions can perform worse than locking
• Simple microbenchmark:

![Graph showing performance comparison between locks and transactions]

• Common optimizations or data structures are not TM-friendly (memoization, counters, linked-lists)
• Limitations of the TM implementation can harm performance
  • For example, overflow in an HTM
• Bobha et al. [ISCA ’07] list other performance pathologies for HTMs

• Lesson: Performance problems in a transactional application can be subtle and counterintuitive

System Effects

TM performance affected by system components
• Compilers such as gcc are optimized to avoid branches on heavily pipelined architectures.
• Example code and (simplified) assembly generated by gcc:

```assembly
if(a < threshold)  
  shared_variable = new_value;

; ebx is new value
cmp 0xc03e6008, %eax
cmperv 0xc03e600c, %edx
mov %edx, 0xc03e600c
```

• Generated code always writes to memory
• Condition determines which value is written back
• Leads to larger performance loss on an HTM than an extra branch penalty

• Lesson: System components such as the compiler, OS, libraries, and microarchitecture can have subtle interactions with TM that lead to performance problems

Syncchar

Syncchar is a model and tool to help programmers understand TM performance
• Syncchar takes a lock-based program and predicts performance under TM
• Pragmatic: most existing parallel applications use locks
• Predictions guide performance tuning
  • Poor performance prediction focuses programmer effort on application restructuring
  • Good performance prediction and bad realized performance indicates TM implementation problem

Formalizes intuition that conflicts should have first-order performance effect
• Data Independent: critical regions cannot conflict
  • Data independent code can scale linearly with CPUs
  • Example: all variables touched are thread-private
• Conflict Density (D): how many critical regions will be involved in a conflict

Ex:

<table>
<thead>
<tr>
<th>Low Density</th>
<th>High Density</th>
</tr>
</thead>
<tbody>
<tr>
<td>Write a</td>
<td>Write a</td>
</tr>
<tr>
<td>Read a</td>
<td>Write a</td>
</tr>
<tr>
<td>Read a</td>
<td>Write a</td>
</tr>
</tbody>
</table>

Time

• More dense conflicts have a longer serial schedule
• Performance predictions based on sampling address sets of critical region and estimating concurrent speedup

Measured vs. Predicted Performance

• Predictions track scaling trends
  • 25% geometric mean error in predictions for STAMP
• Good balance between accuracy and complexity

Full paper to appear at ISPASS 2010