Optimizing Dynamic Race Detection With Hash Consing

David Moon '16 (dm7@williams.edu) and Stephen Freund (freund@cs.williams.edu)
Department of Computer Science at Williams College
August 14, 2015

Multithreaded Programs and Race Conditions

- Multithreaded programs run multiple threads simultaneously, the steps of which are interleaved by a scheduler.
- Each thread can read and write to memory.
- Threads may interfere with each other if they access the same memory location at the “same time”.
- This is called a race condition and causes non-deterministic behavior.

Automatic Dynamic Race Detection

- As the program executes, the detector builds a happens-before graph to capture the relative order of steps taken by different threads.
- This order is determined by the locking operations.
- For example, a release lock operation happens before the next acquire lock operation in the executed interleaving.
- The program may have many other interleavings with the same happens-before graph.
- A race condition between two memory accesses is present if there is no path between them in the graph, which means there exist other interleavings where the accesses occur opposite the observed order.

Our Work: Saving Space with Hash Consing

- Oftentimes many shadow states have the same value.
- The following program execution leaves array element 1 through 99 with the same shadow state value.

A Broken Bank Account and Two Interleavings

<table>
<thead>
<tr>
<th>Deposit:</th>
<th>Good Sched:</th>
<th>Bad Sched:</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1 = bal</td>
<td>bal = 0</td>
<td>bal = 0</td>
</tr>
<tr>
<td>bal = t1+10</td>
<td>t1 = bal</td>
<td>t1 = bal</td>
</tr>
<tr>
<td></td>
<td>bal = t1+10</td>
<td>t2 = bal</td>
</tr>
<tr>
<td>Withdraw:</td>
<td>t2 = bal</td>
<td>bal = t1+10</td>
</tr>
<tr>
<td></td>
<td>bal = t2-10</td>
<td>bal = t2-10</td>
</tr>
<tr>
<td></td>
<td>bal = 0</td>
<td>bal = -10</td>
</tr>
</tbody>
</table>

Locking to Control Scheduling

- Programmers can use locks to avoid bad schedules.
- Locks are objects which can only be held by a single thread at any time, forcing all other threads to wait their turn.

Bank Account with Locking

<table>
<thead>
<tr>
<th>Deposit:</th>
<th>Schedule:</th>
</tr>
</thead>
<tbody>
<tr>
<td>acquire lock</td>
<td>bal = 0</td>
</tr>
<tr>
<td>t1 = bal</td>
<td>acquire lock</td>
</tr>
<tr>
<td>bal = t1+10</td>
<td>t1 = bal</td>
</tr>
<tr>
<td>release lock</td>
<td>bal = t1+10</td>
</tr>
<tr>
<td>Withdraw:</td>
<td>acquire lock</td>
</tr>
<tr>
<td>acquire lock</td>
<td>t2 = bal</td>
</tr>
<tr>
<td>t2 = bal</td>
<td>bal = t2-10</td>
</tr>
<tr>
<td>release lock</td>
<td>bal = 0</td>
</tr>
</tbody>
</table>

Happens-Before Graph & Shadow State

Thread1: Thread2:

1: acquire lock
2: \[x = 1\]
3: release lock
4: acquire lock
5: \[x = 2\]
6: release lock
7: \[x = 3\]

Problem: Lots o’ State

- Unfortunately, the analysis above incurs a large space overhead because it allocates a shadow state object for each memory location.

Future Work

- Assess overall memory savings.
- Hash consing increases time overhead because it requires table lookups and copying of shadow state at each memory access. We plan to optimize our hash consing methods to address this.

--

This research was supported by NSF grant 1421051. Thanks to James Wilcox '13 for help with background material and the poster template.

Preliminary Results

Tests on the Java Grande benchmarks show orders of magnitude reduction in the shadow state size.

Race Conditions Are Still A Problem

- Writing correct multithreaded programs is hard!
- Programmers often forget to acquire the proper locks.
- This can lead to subtle bugs that produce observable error only on certain interleavings, making them hard to detect.