Dynamic Synthesis for Relaxed Memory Models

Feng Liu*, Nayden Nedev*, Nedyalko Prasadnikov†, Martin Vechev§, Eran Yahav‡

*Princeton University, † Sofia University, § ETH Zurich, ‡ Technion

06/13/2012
PLDI 2012, Beijing
Relaxed memory models

• No sequential consistency (SC) in chips today
• Chip designers implement “relaxed memory” on different architectures:

  - total store order (TSO)
    *Intel’s and AMD’s X86; SPARC*

  - partial store order (PSO)
    *SPARC*

  - PPC model
    *IBM’s PowerPC; ARM*

  - ...
Modeling TSO & PSO Programs

• Store Buffering
  – FIFO queues (buffers) associated with threads
  – A store goes to a local buffer, not memory
  – Stores in buffers are flushed at non-deterministic times

• Store Forwarding
  – Satisfy loads from local buffer if possible
PSO Example

```
H=0, Done=0

thread1
H=1;
Done=1;

thread2
while (!Done) {
    assert(H == 1);
}
```

Fails on PSO
Memory Fences

H=0, Done=0

thread1
H=1;
Fence;
Done=1;

thread2
while (!Done) { }
assert(H==1);

Memory fence is very expensive (10-100 cycles)
Use only where necessary
Our Approach

- **C/C++ Program** $P$
- **Specification** $S$
- **Memory Model** $M$

**FENDER**
Dynamic Analysis & static fixing

Program $P'$ with Fences

$P'$ satisfies $S$ under $M$
Challenge: Handling real-world concurrent programs

A lock-free memory allocator

771 lines of C code
2699 lines of IR code

Real-World Programs?

• Exposing violations under relaxed memory models
  – Violations occur rarely

• Many possible fence placements
  – Large programs

• Written in C/C++ language
  – Rather than program models
Contributions

• Demonic scheduler to expose violations
  – Delay flushes of values from store buffer to main memory

• Avoiding bad executions by adding fences
  – Extracting ordering constraints from bad executions
  – Enforcing ordering constraints using fences

• Parametric synthesis framework
  – Different memory models

• Evaluating fences required under different memory models and correctness criteria
  – Found redundant and missing fences
  – Linearizability on relaxed memory models
  – Handled real C/C++ programs
Fender Framework – Support for concurrency and RMM

Concurrent C/C++ code

Client

LLVM-GCC

.thc

LLVM Interpreter

Threading

Demonic Scheduler

Memory Model

our extension

existing work
Our work – Dynamic analysis

Concurrent C/C++ code

Client

LLVM-GCC

.bc

LLVM Interpreter

Threading

Demonic Scheduler

Memory Model

trace

Trace Analysis

Specification

Order formula

SAT Solver

SAT assignment

our extension

existing work
Our work – Implement memory fences

Concurrent C/C++ code

LLVM-GCC

LLVM Interpreter

Threading

Demonic Scheduler

Memory Model

Trace Analysis

Specification

SAT Solver

Fixed bytecode & Fence location report

Concurrent C/C++ code

Client

Trace

Order formula

SAT assignment

our extension

existing work
Example

H=0, Done=0

thread1
L1: H=1;
L2: Done=1;

thread2
L3: while (!Done) { }
L4: assert(H==1);

: 
: 

Main Memory

thread1
H
Done

thread2
H
Done

...
Interpretation on PSO

thread1
L1: H=1;
L2: Done=1;
L3: while (!Done) { }
L4: assert(H==1);
:
:

thread2
H=0, Done=0

trace

L3: Load Done
L1: Store H=1
L2: Store Done=1
L4: Load H

load
store
flush
Interpretation on PSO

```
thread1
L1: H=1;
L2: Done=1;
L3: while (!Done) { }
L4: assert(H==1);
```

```
thread2
L3: while (!Done) { }
L4: assert(H==1);
```

```
H=0, Done=0
```

```
L3
L1
L2
L3
```

```
load
store
flush
```

```
t1
H
Done

H
Done

Main Memory

H
Done

H
Done
```
Flush with a probability

thread1
L1: H=1;
L2: Done=1;
L3: while (!Done) { }
L4: assert(H==1);

thread2
L3: while (!Done) { }
L4: assert(H==1);

H=0, Done=0

trace

Main Memory

load
store
flush

L3
L1
L2
L3
Execution trace

```
H=0, Done=0

thread1
L1: H=1;
L2: Done=1;
L3: while (!Done) { }
L4: assert(H==1);

thread2
L3: while (!Done) { }
L4: assert(H==1);
```

---

**trace**

- **L3**
- **L1**
- **L2**
- **L3**
- **L4**

**load**

- **store**

- **flush**
Repair one trace

order predicate \([L1, L2]\)

order formula \([L1, L2] \lor [C, D] \lor \ldots \text{ for a single execution}\)
Repair all incorrect traces

Global formula to SAT solver:

\[(x_1 \lor x_2 \lor \ldots) \land (x_1 \lor x_3 \lor \ldots) \land \ldots\]

\[\text{trace1} \quad \text{trace3}\]

One memory fence should be placed here
Fix the program

H=0, Done=0

thread1

L1: H=1;
Fence;
L2: Done=1;

thread2

L3: while(! Done) { }
L4: assert(H==1);

....
Evaluation - Benchmarks

Program

<table>
<thead>
<tr>
<th>Program</th>
<th>TSO</th>
<th>PSO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chase-Lev WSQ</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Cilk’s THE WSQ</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>FIFO WSQ</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>LIFO WSQ</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Anchor WSQ</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>FIFO iWSQ</td>
<td>3</td>
<td>-</td>
</tr>
<tr>
<td>LIFO iWSQ</td>
<td>2</td>
<td>-</td>
</tr>
<tr>
<td>Anchor iWSQ</td>
<td>2</td>
<td>-</td>
</tr>
<tr>
<td>MS2 Queue</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>MSN Queue</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>LazyList Set</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Harris’s Set</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Memory allocator</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

- Work stealing queues
- Idempotent Work stealing queues
- Concurrent data structures
- Lock-free memory allocator
## Evaluation - Specifications

<table>
<thead>
<tr>
<th>Program</th>
<th>Memory safety</th>
<th>Operational Sequential consistency</th>
<th>Linearizability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chase-Lev WSQ</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cilk’s THE WSQ</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FIFO WSQ</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LIFO WSQ</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Anchor WSQ</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FIFO iWSQ</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LIFO iWSQ</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Anchor iWSQ</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MS2 Queue</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MSN Queue</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LazyList Set</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Harris’s Set</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Memory allocator</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Evaluation - Memory models

<table>
<thead>
<tr>
<th>Program</th>
<th>Memory safety</th>
<th>Operational Sequential consistency</th>
<th>Linearizability</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>TSO</td>
<td>PSO</td>
<td>TSO</td>
</tr>
<tr>
<td>Chase-Lev WSQ</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cilk’s THE WSQ</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FIFO WSQ</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LIFO WSQ</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Anchor WSQ</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FIFO iWSQ</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LIFO iWSQ</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Anchor iWSQ</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MS2 Queue</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MSN Queue</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LazyList Set</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Harris’s Set</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Memory allocator</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Evaluation - number of memory fences

<table>
<thead>
<tr>
<th>Program</th>
<th>Memory safety</th>
<th>Operational Sequential consistency</th>
<th>linearizability</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>TSO</td>
<td>PSO</td>
<td>TSO</td>
</tr>
<tr>
<td>Chase-Lev WSQ</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Cilk’s THE WSQ</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>FIFO WSQ</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>LIFO WSQ</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Anchor WSQ</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>FIFO iWSQ</td>
<td>0</td>
<td>3</td>
<td>-</td>
</tr>
<tr>
<td>LIFO iWSQ</td>
<td>0</td>
<td>2</td>
<td>-</td>
</tr>
<tr>
<td>Anchor iWSQ</td>
<td>0</td>
<td>2</td>
<td>-</td>
</tr>
<tr>
<td>MS2 Queue</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>MSN Queue</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>LazyList Set</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Harris’s Set</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Memory allocator</td>
<td>0</td>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>
Conclusion

• Demonic scheduler to expose violations

• Avoiding bad executions by adding fences

• Parametric synthesis framework

• Evaluating fences required under different memory models and correctness criteria
Thanks!

Q & A