Don’t Sit on the Fence
A Static Analysis Approach to Automatic Fence Insertion

Or Ostrovsky

April 25th 2018
Table of contents

1 Introduction
2 Cycles
3 Static Analysis
4 Soundness of Construction
5 Fence placement
6 Conclusion
Table of contents

1. Introduction
2. Cycles
3. Static Analysis
4. Soundness of Construction
5. Fence placement
6. Conclusion
Problem

- Programming not under SC is complicated
- Programmers are stupid
Problem

- Programming not under SC is complicated
- Programmers are stupid
- Solution: Let the computer do it
Problem

- Programming not under SC is complicated
- Programmers are stupid
- Solution: Let the computer do it
- Easier said than done
Goal

- Simulate Sequential Consistency, using fences
- Automatic
- Optimal
Challenges
Challenges

- Correctness
- Optimality
- Scalability
- Compiler optimizations
Table of contents

1 Introduction

2 Cycles

3 Static Analysis

4 Soundness of Construction

5 Fence placement

6 Conclusion
Memory models: Recap.

- Operational vs. Axiomatic
- Different relations
  - Program Order \((po)\)
  - Coherence \((co)\)/Memory Order \((mo)\)
  - Read From \((rf)\)
  - From Read \((fr = rf^{-1}; co)\)
  - Static vs. dynamic
- Sequential Consistency vs. Relaxed memory models
  - SC: \(acyclic(po \cup co \cup rf \cup fr)\)
  - Relaxed: only a subset
Candidate execution

**Definition**

<table>
<thead>
<tr>
<th>Event</th>
<th>$W_{xv}, R_{xv}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event Structure</td>
<td>$E \triangleq (E, po), E = {\text{events}}$</td>
</tr>
<tr>
<td>Execution Witness</td>
<td>$X \triangleq (co, rf, fr)$</td>
</tr>
<tr>
<td>Candidate Execution</td>
<td>$(E, X)$</td>
</tr>
<tr>
<td>Memory Model</td>
<td>$MM : {(E, X)} \mapsto {\text{true, false}}$</td>
</tr>
</tbody>
</table>
Candidate execution

Definition

Event \( W_{xv}, R_{xv} \),

Event Structure \( E \triangleq (\mathcal{E}, po), \mathcal{E} = \{events\} \),

Execution Witness \( X \triangleq (co, rf, fr) \),

Candidate Execution \( (E, X) \),

Memory Model \( MM : \{(E, X)\} \mapsto \{true, false\} \),

- Construction?
Candidate execution

Example

(a) Wx1  (c) Ry1
   po   po
  ↓   ↓
(b) Wy1 (d) Rx0

(a) event structure  (b) candidate execution

(a) Wx1
   fr
  ↓
(b) Wy1

(c) Ry1
   rf
  ↓
(d) Rx0
   po
Minimal cycles

Definition

**MC1** Per thread:
- At most 2 accesses
- Accesses are adjacent in the cycle

**MC2** Per memory location:
- At most 3 accesses
- Accesses are adjacent in the cycle
Minimality condition: MC2

Example

(d) Rx2 ← ↫ (c) Wx2

... ↫ ... ↫ ...

(a) Wx1 → ↫ (b) Rx1

(a) cycle

(b) shortcut in cycle

(d) Rx2 ← ↫ (c) Wx2

... ↫ ... ↫ ...

(a) Wx1 → ↫ (b) Rx1

(c) shortcut in cycle
Delay cycles

**Definition**

Delay is a relaxed edge of $p_0$, or $rf$ on an architecture $A$ (MM). Delays can be prevented using fences.

**Theorem**

A *candidate execution* is valid on $A$ but not on SC if:

- **DC1** It contains at least one cycle that has a delay.
- **DC2** All of the cycles contain a delay.
Critical cycles

Definition

**CS1** At least one delay

**CS2** Per thread:
- At most 2 accesses
- Accesses are adjacent in the cycle
- To different memory locations

**CS3** Per memory location:
- At most 3 accesses
- Accesses are adjacent in the cycle
- From different threads
Critical cycles

**Definition**

**CS1** At least one delay

**CS2** Per thread:
- At most 2 accesses
- Accesses are adjacent in the cycle
- To different memory locations

**CS3** Per memory location:
- At most 3 accesses
- Accesses are adjacent in the cycle
- From different threads
Critical cycles: proof

**Theorem**

*If an execution candidate is valid on A but not on SC, then there is a cycle which satisfies:*

1. Is a minimal cycle.
2. Has least one delay.
3. Accesses on the same threads are to different locations
4. Accesses to the same location are from different threads*
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Introduction</td>
</tr>
<tr>
<td>2</td>
<td>Cycles</td>
</tr>
<tr>
<td>3</td>
<td>Static Analysis</td>
</tr>
<tr>
<td>4</td>
<td>Soundness of Construction</td>
</tr>
<tr>
<td>5</td>
<td>Fence placement</td>
</tr>
<tr>
<td>6</td>
<td>Conclusion</td>
</tr>
</tbody>
</table>
Abstract Event Graph

**Definition**

- **Abstract Event** $W_x, R_x$: Abstraction of events
- **Static event set** $\mathbb{E}_s = \{abstract\ events\}$
- **Static Program Order** $p_o_s$: Abstraction of po
- **Competing pairs** $cmp$: Communication between threads
- **AEG** $aeg \triangleq (\mathbb{E}_s, p_o_s, cmp)$
Abstract Event Graph

Example

(a) Wx1  
(b) Wy1  
(c) Ry1  
(d) Rx0

po  fr  rf  po

(a) Wx  
(b) Wy  
(c) Ry  
(d) Rx

cmp  cmp  po_s
AEG construction

• Convert C program to “goto-instructions”
• Ignore local variables
• Read each instruction, and update the AEG, starting from the empty graph.
• Semi-formally:

\[ \tau[i_k; \ldots](aeg) = \tau[i_k'; \ldots](f(aeg, (i_k, \ldots, i_{k' - 1}))) \]
Goto instructions

Example

```c
void thread_1(int input) {
    int r1;
    x = input;
    if (rand()%2)
        y = 1;
    else
        r1 = z;
    x = 1;
}
```

```c
void thread_2() {
    int r2, r3, r4;
    r2 = y;
    r3 = z;
    r4 = x;
}
```

```c
thread_1
    int r1;
    x = input;
    _Bool tmp;
    tmp = rand();
    ![tmp%2] goto 1;
    y = 1;
    goto 2;
1: r1 = z;
2: x = 1;
end_function
```

```c
thread_2
    int r2, r3, r4;
    r2 = y;
    r3 = z;
    r4 = x;
end_function
```
Transformation function

Example

\[ \tau[x = f(y_1, \ldots, y_k); i](E_s, po_s, cmp) = \]
\[ \text{let } reads = \{Ry_1, \ldots, Ry_k\} \text{ in} \]
\[ \text{let } writes = \{Wx\} \text{ in} \]
\[ \text{let } E'_s = E_s \cup reads \cup writes \text{ in} \]
\[ \text{let } po'_s = po_s \cup (end(po_s) \times reads) \cup (reads \times writes) \text{ in} \]
\[ \tau[i](E'_s, po'_s, cmp) \]

end(x) all sink events of x
Transformation function: cont.

Example

\[ \tau[\text{start\_thread } \text{th}; i](\text{aeg}) = \]

\[
\text{let } \text{main } = \tau[\text{body}(\text{th})](\emptyset) \text{ in } \\
\text{let } \text{local } = \tau[i](\text{aeg}) \text{ in } \\
\text{let } \text{inter } = \tau[i](\emptyset) \text{ in }
\]

\[
(\text{local.\text{Es}} \cup \text{main.\text{Es}}, \text{local.\text{po}}_s \cup \text{main.\text{po}}_s, \text{local.\text{Es}} \otimes \text{inter.\text{Es}})
\]

\[
A \otimes B \triangleq \{(a, b) \in A \times B | \\
\text{addr}(a) = \text{addr}(b) \land \\
(\text{write}(a) \lor \text{write}(b))\}
\]

\[
\emptyset \triangleq (\emptyset, \emptyset, \emptyset)
\]
### Example

thread_1

```c
int r1;
_Bool tmp;
tmp = rand();
[!tmp%2] goto 1;
```

1. `x = input;`
2. `y = 1;`
3. `r1 = z;`
4. `x = 1;`

end_function

thread_2

```c
int r2, r3, r4;
r2 = y;
r3 = z;
r4 = x;
```

5. `x = input;`
6. `y = 1;`
7. `r1 = z;`
8. `x = 1;`

end_function

---

(a) \text{Wx} 
(b) \text{Rz} 
(c) \text{Wx} 
(d) \text{Ry} 
(e) \text{Rz} 
(f) \text{Rx} 

\text{cmp} \quad \text{cmp} \quad \text{cmp} 

\text{po}_s \quad \text{po}_s \quad \text{po}_s 

Don't Sit on the Fence
Event structure construction

- Analogous to AEG
- \( S(P) = \{(E, po)\} \): possible event structures
- \( S(P) = \sigma(P)(\emptyset) \): \( \sigma \) is very much like \( \tau \)
Transformation function

Example

\[ \sigma[lhs = rhs; i](ses) = \]

\[
\begin{align*}
\text{let } de & = \text{dyn\_evts}(lhs = rhs) \text{ in} \\
\text{let } E'(E, w, R) & = E \cup \{w\} \cup R \text{ in} \\
\text{let } po'(po, w, R) & = po \cup (end(po) \times R) \cup (R \times \{w\}) \text{ in} \\
\text{let } es'(es, w, R) & = (E'(es.E, w, R), po'(es.po, w, R)) \text{ in} \\
\sigma[i](\{es'(es, w, R) | es \in ses, (w, R) \in de\})
\end{align*}
\]

- \( \text{dyn\_evts}(lhs = rhs) = \{(w, R)\} \):
  - Set of events that can cause the statement.
  - Example:
    \[
    \text{dyn\_evts}(x = y + z) = \bigcup\{(Wxv_1, \{Ryv_2, Rzv_3\}) | v_1 = v_2 + v_3\}
    \]
Example

\[ \sigma[\text{start\_thread th; i}](\text{ses}) = \]

\[
\text{let local } = \sigma[\text{body(th)}](\emptyset) \text{ in }
\]

\[
\text{let main } = \sigma[i](\text{ses}) \text{ in }
\]

\[
\bigcup_{es_l \in \text{local}, es_m \in \text{main}} \{(es_l.\mathcal{E} \cup es_m.\mathcal{E}, \text{es}_l.\text{po} \cup \text{es}_m.\text{po})\}
\]
Example

(a) aeg of Figure 9

(b) ex. with critical cycle

(c) ex. without critical cycle
Loops

- Event $a$ might depend on itself on previous iterations
- In that case, duplicate loop body
Table of contents

1. Introduction
2. Cycles
3. Static Analysis
4. Soundness of Construction
5. Fence placement
6. Conclusion
Soundness

- $G = aeg(P)$
- $E \in S(P)$
- Are they related?
Concretization

Definition

\[ \gamma_e(se) \triangleq \{ e' \mid \exists e \in se \text{ s.t. } addr(e) = addr(e') \land \]
\[ \text{dir}(e) = \text{dir}(e') \land \text{origin}(e) = \text{origin}(e') \} \]

\[ \gamma(srel) \triangleq \{(c_1, c_2) \mid \exists (s_1, s_2) \in srel \text{ s.t.} \]
\[ (c_1, c_2) \in \gamma_e(\{s_1\}) \times \gamma_e(\{s_2\}) \} \]

Theorem

\[ E_1 \subseteq \gamma_e(E_{s,1}), E_2 \subseteq \gamma_e(E_{s,2}) \Rightarrow E_1 \times E_2 \subseteq \gamma(E_{s,1} \times E_{s,2}) \]
Events and program order

Theorem

\[ E \in S(P), \ G = aeg(P) \Rightarrow E.E \subseteq \gamma_e(G.E_s), \ E.po \subseteq \gamma(G.po_s^+) \]

- Lemma 5.3 in the article
- po\(^+\) is po’s closure
rf, co, and fr

**Theorem**

\[ E \in S(P), X = (rf, co, fr), (E, X) \text{ is a CE}, G = aeg(P) \]
\[ \implies X.rfe, X.coe, X.fre \subseteq \gamma(G.cmp) \]

- Lemma 5.4 in the article
Soundness

Theorem

Let $P$ be a program. Let $E \in S(P)$, $X = (\text{rf}, \text{co}, \text{fr})$ an execution witness, $(E, X)$ a candidate execution. Also, let $G = \text{aeg}(P)$.

$$E.\text{po} \cup X.\text{coi} \cup X.\text{rfi} \cup X.\text{fri} \subseteq \gamma(G.\text{po}_s^+)$$

$$X.\text{coe} \cup X.\text{rfe} \cup X.\text{fre} \subseteq \gamma(G.\text{cmp})$$

$$E.\mathbb{E} \subseteq \gamma_e(G.\mathbb{E}_s)$$

- From the two previous theorems
Static critical cycles

Theorem

Let $E \in S(P)$, $X = (rf, co, fr)$, $G = aeg(P)$. If $(E, X)$ contains a critical cycle $c = c_0, \ldots, c_{n-1}$, then there is a cycle $d = d_0, \ldots, d_{n-1}$ in $G$ so that:

- $\{c_i\} \subseteq \gamma_e(\{d_i\})$
- $\{(c_i, c_{i+1} \mod n)\} \subseteq \gamma(\{(d_i, d_{i+1} \mod n)\})$

- Looking for cycles in $G$ will find all cycles in $(E, X)$
- Any cycle detection algorithm will do.
Static critical cycles

Example

\begin{align*}
(a) & W_x \\
(b_2) & R_z \\
(c) & W_x \\
(d) & R_y \\
(e) & R_z \\
(f) & R_x
\end{align*}

\begin{align*}
po_s & \xrightarrow{cmp} po_s \\
po_s & \xrightarrow{cmp} po_s \\
po_s & \xrightarrow{cmp} po_s \\
po_s & \xrightarrow{cmp} po_s \\
po_s & \xrightarrow{cmp} po_s \\
po_s & \xrightarrow{cmp} po_s
\end{align*}

\begin{align*}
(a') & W_{x1} \\
(b'_2) & R_z \\
(c') & W_{x1} \\
(d') & R_{y1} \\
(e') & R_{z0} \\
(f') & R_{x0}
\end{align*}

\begin{align*}
po & \xrightarrow{rf} po \\
po & \xrightarrow{fr} po \\
po & \xrightarrow{fr} po \\
po & \xrightarrow{fr} po \\
po & \xrightarrow{fr} po \\
po & \xrightarrow{fr} po
\end{align*}
Static critical cycles

Example

- $a'$, $b'_1$, $d'$, $e'$, $f'$
# Table of contents

1. Introduction
2. Cycles
3. Static Analysis
4. Soundness of Construction
5. Fence placement
6. Conclusion
Considerations

- We have a list of cycles $C = \{C_1, \ldots, C_n\}$. Now what?
Considerations

- We have a list of cycles $C = \{C_1, \ldots, C_n\}$. Now what?
- Delays
- Fence types, locations & costs
- Different for each architecture
Problem parameters

• Input:
  ▶ $aeg(E_s, p_o, cmp)$
  ▶ $C = \{ C_1, \ldots, C_n \}$
  ▶ $T = \{ f, lwf, cf, dp \}$, $cost : T \rightarrow \mathbb{N}$
  ▶ $placements(C) \subseteq p_o \times T$
  ▶ Constrain $^1$

• Output:
  ▶ $\forall (l, t) \in placements(C), t_l \in \{0, 1\}$

• Cost function:
  ▶ Rough estimation of $cost$
  ▶ Minimize $\sum_{(l, t) \in placements(C)} t_l \times cost(t)$
  ▶ Problems?

$^1$ Architecture dependent
Constraints

- Every delay needs to be fenced
- Each type of delay can be handled by different types of fences
- A fence can “participate” in multiple delays
- “Any of” condition: \( \ldots \geq 1 \)
  - Promises the problem is satisfiable
  - Trust the cost function
TSO delays & fences

- One type of fence £
- Only poWR delays
AEG in TSO

Example

\[(a)Wt \xRightarrow{p_{os}} (b)Wy\]
\[(c)Rz \xRightarrow{p_{os}} (d)Wx\]
\[(e)Rx \xRightarrow{p_{os}} (f)Ry\]
\[(g)Wz \xRightarrow{f_{os}} (h)Rt\]
\[(i)Rz \xRightarrow{p_{os}} (j)Wy\]
\[(k)Wt \xRightarrow{f_{os}} (l)Rz\]
Not that bad, right?
Delays  poWR, poWW, poRW, poRR

\begin{itemize}
  \item \textit{f} Can solve delays in $po_s^+$. \hfill
  \[
  between(x, y) \triangleq \{(e_1, e_2) \in po_s \mid (x, e_1), (e_2, y) \in po_s^*}\]
  \hfill
  \item \textit{lwf} Same as \textit{f}, but unsuitable for poWR violations.
  \item \textit{dp} Applies only to delays in $po_s$
\end{itemize}

\ldots \ldots
Power: placement & constraints

- Exact definition of placements($C$):
  \[ \text{placements}(C) \triangleq \{(l, dp) | l \in \text{delays}(C)\} \cup \{(l, t) | t \in \mathbb{T} \setminus \{dp\}, \]
  \[ l \in \text{between}(\text{delays}(C))\} \cup \{(l, t) | t \in \{f, lwf\}, l \in \text{po}_s(C)\} \]

- For each $d \in \text{delays}(C)$
  - If $d \in \text{poWR}$ then $\sum_{e \in \text{between}(d)} f_e \geq 1$
  - If $d \in \text{poWW}$ then $\sum_{e \in \text{between}(d)} (f_e + lwf_e) \geq 1$
  - If $d \in \text{poRW} \cup \text{poRR}$ then $dp_d + \sum_{e \in \text{between}(d)} (f_e + lwf_e) \geq 1$
  - ...
Power: placement & constraints

- Exact definition of placements$(C)$:
  \[
  \text{placements}(C) \triangleq \{(l, dp) | l \in \text{delays}(C)\} \cup \\
  \{(l, t) | t \in \mathbb{T} \setminus \{dp\}, \ l \in \text{between}(\text{delays}(C))\} \cup \\
  \{(l, t) | t \in \{f, lwf\}, l \in \text{pos}(C)\}
  \]

- For each $d \in \text{delays}(C)$
  - If $d \in \text{poWR}$ then $\sum_{e \in \text{between}(d)} f_e \geq 1$
  - If $d \in \text{poWW}$ then $\sum_{e \in \text{between}(d)} (f_e + lwf_e) \geq 1$
  - If $d \in \text{poRW} \cup \text{poRR}$ then $dp_d + \sum_{e \in \text{between}(d)} (f_e + lwf_e) \geq 1$
  - ...  

- How to solve? ILP
Example

\begin{align*}
\text{min} & \quad d_p(e,g) + d_p(f,h) + d_p(f,g) + 3 \cdot (f(e,f) + f(f,g) + f(g,h)) \\
& \quad + 2 \cdot (lwf(e,f) + lwf(f,g) + lwf(g,h)) \\
\text{s.t.} & \quad \text{cycle 1, delay } (e,g): \quad d_p(e,g) + f(e,f) + f(f,g) + lwf(e,f) + lwf(f,g) \geq 1 \\
& \quad \text{cycle 2, delay } (f,g): \quad d_p(f,g) + f(f,g) + lwf(f,g) \geq 1 \\
& \quad \text{cycle 3, delay } (f,h): \quad d_p(f,h) + f(f,g) + f(g,h) + lwf(f,g) + lwf(g,h) \geq 1 \\
& \quad \text{cycle 4, delay } (g,h): \quad f(g,h) \geq 1
\end{align*}
Table of contents

1 Introduction

2 Cycles

3 Static Analysis

4 Soundness of Construction

5 Fence placement

6 Conclusion
Evaluation

- Measure how well did we do?
Evaluation

- Measure how well did we do?
- Relative overhead
- Compared to other tools
- Different architectures
Evaluation

- Measure how well did we do?
- Relative overhead
- Compared to other tools
- Different architectures

**Musketeer, Pensieve, Visual Studio, after Each access, after Heap accesses**
Conclusion

- Define critical cycles
- Discover them using static analysis
- Prove the static analysis is sound
- Find the best way to place fences
Excluded topics

- Related works
- Pointer analysis
- Most of the conversion technicalities
- Some architecture specifics
- Implementation & performance (mostly)
Questions?