Don't Sit on the Fence

A Static Analysis Approach to Automatic Fence Insertion

Or Ostrovsky

April 25th 2018

Table of contents

Table of contents

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

Memory models: Recap.

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

Candidate execution

Definition

Event	Wxv, Rxv
Event Structure	$E \triangleq (\mathbb{E}, \mathtt{po}),\ \mathbb{E} = \{events\}$
Execution Witness	$X \triangleq (\mathtt{co}, \mathtt{rf}, \mathtt{fr})$
Candidate Execution	(E, X)
Memory Model	$MM : \{(E, X)\} \mapsto \{true, false\}$

Candidate execution

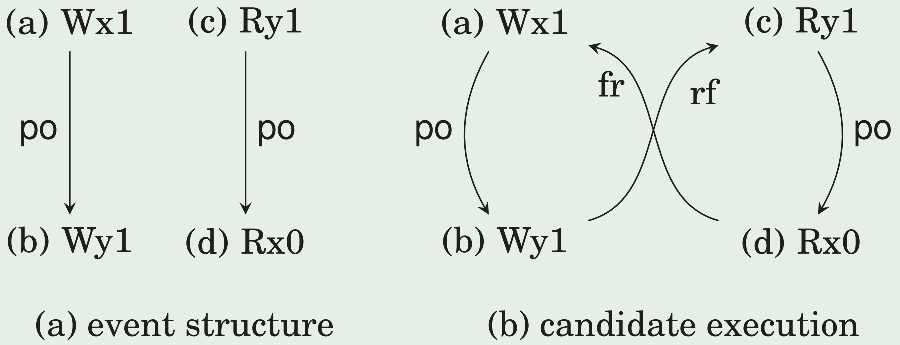
Definition

Event	Wxv, Rxv
Event Structure	$E \triangleq (\mathbb{E}, \mathtt{po}),\ \mathbb{E} = \{events\}$
Execution Witness	$X \triangleq (\mathtt{co}, \mathtt{rf}, \mathtt{fr})$
Candidate Execution	(E, X)
Memory Model	$MM : \{(E, X)\} \mapsto \{true, false\}$

- Construction?

Candidate execution

Example



Minimal cycles

Definition

MC1 Per thread:

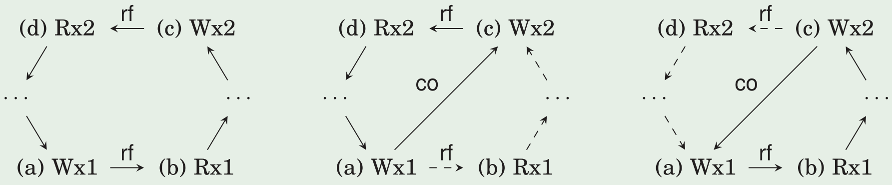
- At most 2 accesses
- Accesses are adjacent in the cycle

MC2 Per memory location:

- At most 3 accesses
- Accesses are adjacent in the cycle

Minimality condition: MC2

Example



(a) cycle

(b) shortcut in cycle

(c) shortcut in cycle

Delay cycles

Definition

Delay is a relaxed edge of `po`, 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 of contents

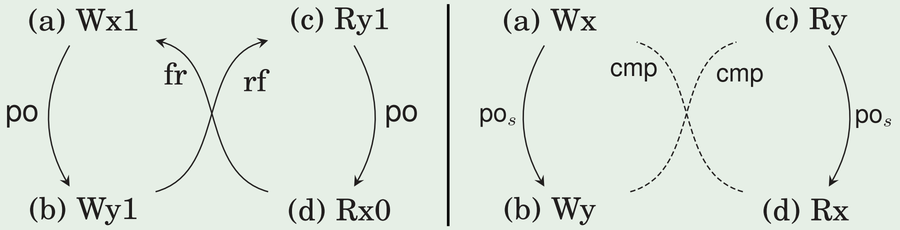
Abstract Event Graph

Definition

Abstract Event	Wx, Rx: Abstraction of events
Static event set	$\mathbb{E}_s = \{abstract\ events\}$
Static Program Order	$\mathtt{po}_s$: Abstraction of po
Competing pairs	$\mathtt{cmp}$: Communication between threads
AEG	$aeg \triangleq (\mathbb{E}_s, \mathtt{po}_s, \mathtt{cmp})$

Abstract Event Graph

Example



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

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

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

```
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
```

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

Transformation function

Example

$$\begin{array}{l}
\tau[x = f(y_1, \ldots, y_k); i](\mathbb{E}_s, \mathrm{po}_s, \mathtt{cmp}) = \\
\quad \textbf{let } \mathit{reads} \quad = \quad \{Ry_1, \ldots, Ry_k\} \textbf{ in} \\
\quad \textbf{let } \mathit{writes} \quad = \quad \{Wx\} \textbf{ in} \\
\quad \textbf{let } \mathbb{E}'_s \quad = \quad \mathbb{E}_s \cup \mathit{reads} \cup \mathit{writes} \textbf{ in} \\
\quad \textbf{let } \mathrm{po}'_s \quad = \quad \mathrm{po}_s \cup (\mathit{end}(\mathrm{po}_s) \times \mathit{reads}) \cup (\mathit{reads} \times \mathit{writes}) \textbf{ in} \\
\tau[i](\mathbb{E}'_s, \mathrm{po}'_s, \mathtt{cmp})
\end{array}$$

$\mathit{end}(x)$ all sink events of x

Transformation function: cont.

Example

$$
\begin{array}{l}
\tau[start\_thread\ \ th; i](aeg) = \\
\quad \textbf{let } main \;=\; \tau[body(th)](\bar{\emptyset}) \textbf{ in} \\
\quad \textbf{let } local \;=\; \tau[i](aeg) \textbf{ in} \\
\quad \textbf{let } inter \;=\; \tau[i](\bar{\emptyset}) \textbf{ in} \\
(local.\mathbb{E}_s \cup main.\mathbb{E}_s, local.\mathrm{po}_s \cup main.\mathrm{po}_s, local.\mathbb{E}_s \otimes inter.\mathbb{E}_s)
\end{array}
$$

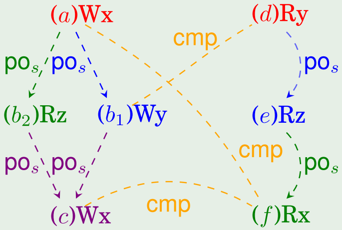
$$
\begin{aligned}
A \otimes B \triangleq & \{(a, b) \in A \times B | \\
& addr(a) = addr(b) \wedge \\
& (write(a) \vee write(b))\} \\
\bar{\emptyset} \triangleq & (\emptyset, \emptyset, \emptyset)
\end{aligned}
$$

Program & AEG

Example

```
  thread_1
     int r1;
1    x = input;
     _Bool tmp;
     tmp = rand();
     [!tmp%2] goto 1;
2    y = 1;
     goto 2;
3 1: r1 = z;
4 2: x = 1;
     end_function
```

```
8 thread_2
     int r2, r3, r4;
5    r2 = y;
6    r3 = z;
7    r4 = x;
     end_function
```



Event structure construction

- Analogous to AEG
- $S(P) = \{(\mathbb{E}, \mathrm{po})\}$: possible event structures
- $S(P) = \sigma(P)(\emptyset)$: σ is very much like τ

Transformation function

Example

$$
\begin{array}{lcl}
\multicolumn{3}{l}{\sigma[lhs = rhs; i](ses) =} \\
\quad \textbf{let } de & = & \texttt{dyn\_evts}(lhs = rhs) \textbf{ in} \\
\quad \textbf{let } \mathbb{E}'(\mathbb{E}, w, R) & = & \mathbb{E} \cup \{w\} \cup R \textbf{ in} \\
\quad \textbf{let } \mathrm{po}'(\mathrm{po}, w, R) & = & \mathrm{po} \cup (end(\mathrm{po}) \times R) \cup (R \times \{w\}) \textbf{ in} \\
\quad \textbf{let } es'(es, w, R) & = & (\mathbb{E}'(es.\mathbb{E}, w, R), \mathrm{po}'(es.\mathrm{po}, w, R)) \textbf{ in} \\
\multicolumn{3}{l}{\sigma[i](\{es'(es, w, R) \mid es \in ses, (w, R) \in de\})}
\end{array}
$$

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

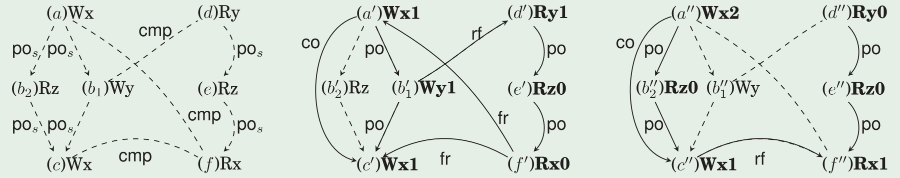
Transformation function: cont.

Example

$$
\begin{aligned}
&\sigma[\mathit{start\_thread}\ \ th; i](ses) = \\
&\ \ \textbf{let}\ \mathit{local} \ \ = \ \ \sigma[\mathit{body}(th)](\emptyset)\ \textbf{in} \\
&\ \ \textbf{let}\ \mathit{main} \ \ = \ \ \sigma[i](ses)\ \textbf{in} \\
&\bigcup_{es_l \in \mathit{local}, es_m \in \mathit{main}} \{(es_l.\mathbb{E} \cup es_m.\mathbb{E}, es_l.\mathrm{po} \cup es_m.\mathrm{po})\}
\end{aligned}
$$

AEG & ES

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

Soundness

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

Concretization

Definition

$$\gamma_e(se) \triangleq \{e' | \exists e \in se \ s.t. \ addr(e) = addr(e') \wedge$$
$$dir(e) = dir(e') \wedge origin(e) = origin(e')\}$$
$$\gamma(srel) \triangleq \{(c_1, c_2) | \exists (s_1, s_2) \in srel \ s.t.$$
$$(c_1, c_2) \in \gamma_e(\{s_1\}) \times \gamma_e(\{s_2\})\}$$

Theorem

$$\mathbb{E}_1 \subseteq \gamma_e(\mathbb{E}_{s,1}), \mathbb{E}_2 \subseteq \gamma_e(\mathbb{E}_{s,2}) \Rightarrow \mathbb{E}_1 \times \mathbb{E}_2 \subseteq \gamma(\mathbb{E}_{s,1} \times \mathbb{E}_{s,2})$$

Events and program order

Theorem

$$E \in S(P), G = aeg(P) \Rightarrow E.\mathbb{E} \subseteq \gamma_e(G.\mathbb{E}_s), E.\mathrm{po} \subseteq \gamma(G.\mathrm{po}_s^+)$$

- Lemma 5.3 in the article
- po^+ is po's closure

$\texttt{rf}$, $\texttt{co}$, and $\texttt{fr}$

Theorem

$$E \in S(P), X = (\texttt{rf}, \texttt{co}, \texttt{fr}), (E, X) \textit{ is a } CE, G = aeg(P)$$
$$\Rightarrow$$
$$\texttt{X.rfe}, \texttt{X.coe}, \texttt{X.fre} \subseteq \gamma(G.\texttt{cmp})$$

- Lemma 5.4 in the article

Soundness

Theorem

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

$$E.\mathtt{po} \cup X.\mathtt{coi} \cup X.\mathtt{rfi} \cup X.\mathtt{fri} \subseteq \gamma(G.\mathtt{po}_s^+)$$
$$X.\mathtt{coe} \cup X.\mathtt{rfe} \cup X.\mathtt{fre} \subseteq \gamma(G.\mathtt{cmp})$$
$$E.\mathbb{E} \subseteq \gamma_e(G.\mathbb{E}_s)$$

- From the two previous theorems

Static critical cycles

Theorem

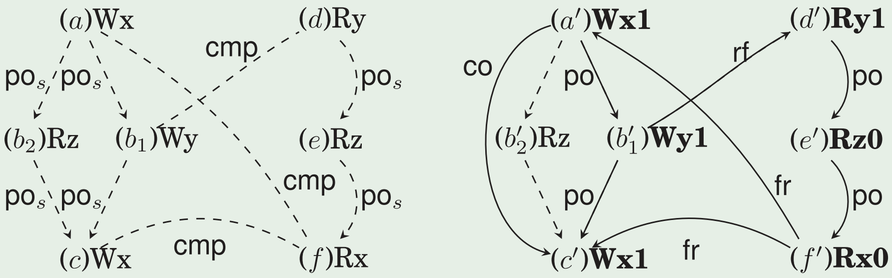
Let $E \in S(P)$, $X = (\texttt{rf}, \texttt{co}, \texttt{fr})$, $G = aeg(P)$. If (E, X) contains a critical cycle $c = c_0, \ldots, c_{n-1}$, then there is a cycle $d = d_0, \ldots, d_{n-1}$ in G so that:

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

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

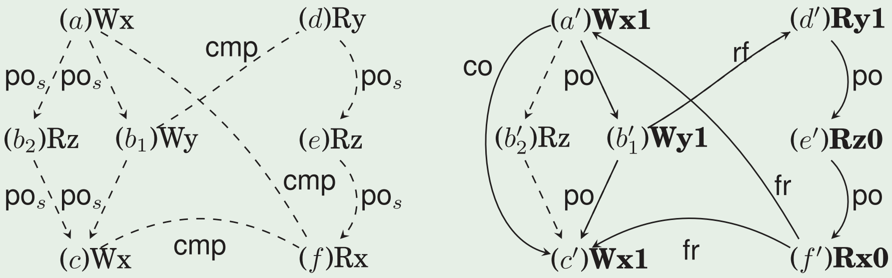
Static critical cycles

Example



Static critical cycles

Example



- a', b'_1, d', e', f'

Table of contents

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:
  - $aeg(\mathbb{E}_s, \texttt{po}_s, \texttt{cmp})$
  - $C = \{C_1, \ldots, C_n\}$
  - $\mathbb{T} = \{\texttt{f}, \texttt{lwf}, \texttt{cf}, \texttt{dp}\}$, $cost : \mathbb{T} \mapsto \mathbb{N}$ [1]
  - $placements(C) \subseteq \texttt{po}_s \times \mathbb{T}$ [1]
  - Constrains [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

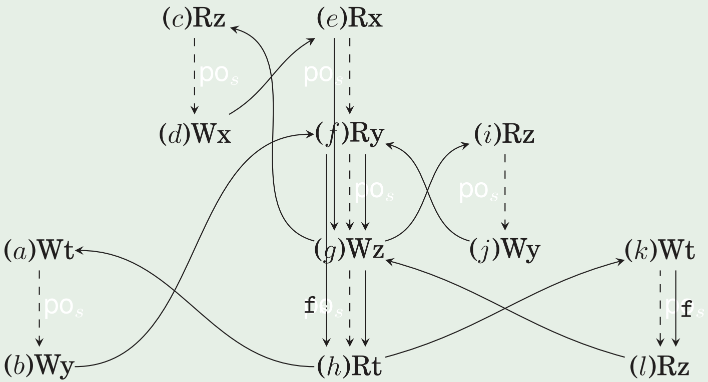
- Every delay needs to be fenced
- Each type of delay can be handled by different types of fences
- A fence can “participate” in multiple delays
- “Any of” condition: $\ldots \geq 1$
  - Promises the problem is satisfiable
  - Trust the cost function

TSO delays & fences

- One type of fence `f`
- Only poWR delays

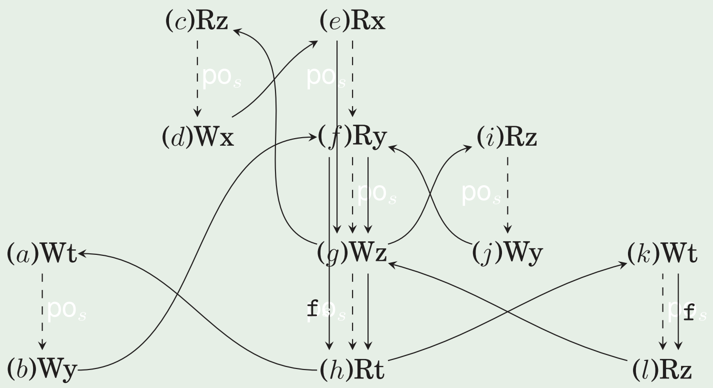
AEG in TSO

Example



AEG in TSO

Example



- Not that bad, right?

Power: delays & fences

Delays poWR, poWW, poRW, poRR

$\mathtt{f}$ Can solve delays in $\mathtt{po}_s^+$.

$$between(x, y) \triangleq \{(e_1, e_2) \in \mathtt{po}_s \mid (x, e_1), (e_2, y) \in \mathtt{po}_s^*\}$$

$\mathtt{lwf}$ Same as $\mathtt{f}$, but unsuitable for poWR violations.

$\mathtt{dp}$ Applies only to delays in $\mathtt{po}_s$

... ...

Power: placement & constraints

- Exact definition of *placements*(C):

$$
\begin{aligned}
placements(C) \triangleq & \{(l, \mathtt{dp}) | l \in delays(C)\} \cup \\
& \{(l, t) | t \in \mathbb{T} \setminus \{\mathtt{dp}\}, \\
& l \in between(delays(C))\} \cup \\
& \{(l, t) | t \in \{\mathtt{f}, \mathtt{lwf}\}, l \in \mathtt{po}_s(C)\}
\end{aligned}
$$

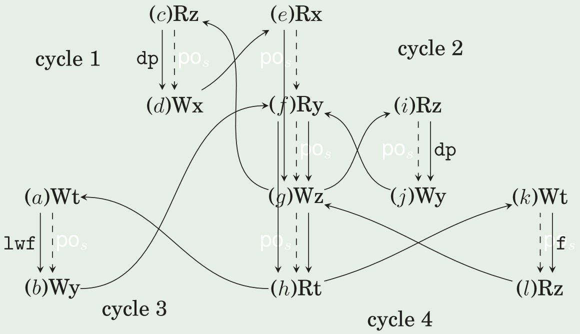
- For each $d \in delays(C)$
  - If $d \in poWR$ then $\sum_{e \in between(d)} f_e \geq 1$
  - If $d \in poWW$ then $\sum_{e \in between(d)} (f_e + lwf_e) \geq 1$
  - If $d \in poRW \cup poRR$ then $dp_d + \sum_{e \in between(d)} (f_e + lwf_e) \geq 1$
  - ...

Power: placement & constraints

- Exact definition of *placements*(C):
$$
\begin{aligned}
placements(C) \triangleq & \{(l, \mathtt{dp}) | l \in delays(C)\} \cup \\
& \{(l, t) | t \in \mathbb{T} \setminus \{\mathtt{dp}\}, \\
& l \in between(delays(C))\} \cup \\
& \{(l, t) | t \in \{\mathtt{f}, \mathtt{lwf}\}, l \in \mathtt{po}_s(C)\}
\end{aligned}
$$
- For each $d \in delays(C)$
  - If $d \in poWR$ then $\sum_{e \in between(d)} f_e \geq 1$
  - If $d \in poWW$ then $\sum_{e \in between(d)} (f_e + lwf_e) \geq 1$
  - If $d \in poRW \cup poRR$ then $dp_d + \sum_{e \in between(d)} (f_e + lwf_e) \geq 1$
  - ...
- How to solve? ILP

AEG & ILP

Example



$$
\begin{array}{ll}
\textbf{min} & \mathtt{dp}_{(e,g)} + \mathtt{dp}_{(f,h)} + \mathtt{dp}_{(f,g)} + 3 \cdot (\mathtt{f}_{(e,f)} + \mathtt{f}_{(f,g)} + \mathtt{f}_{(g,h)}) \\
 & +2 \cdot (\mathtt{lwf}_{(e,f)} + \mathtt{lwf}_{(f,g)} + \mathtt{lwf}_{(g,h)}) \\
\textbf{s.t.} & \text{cycle 1, delay } (e,g)\text{:}\ \ \mathtt{dp}_{(e,g)} + \mathtt{f}_{(e,f)} + \mathtt{f}_{(f,g)} + \mathtt{lwf}_{(e,f)} + \mathtt{lwf}_{(f,g)} \geq 1 \\
 & \text{cycle 2, delay } (f,g)\text{:}\ \ \mathtt{dp}_{(f,g)} + \mathtt{f}_{(f,g)} + \mathtt{lwf}_{(f,g)} \geq 1 \\
 & \text{cycle 3, delay } (f,h)\text{:}\ \ \mathtt{dp}_{(f,h)} + \mathtt{f}_{(f,g)} + \mathtt{f}_{(g,h)} + \mathtt{lwf}_{(f,g)} + \mathtt{lwf}_{(g,h)} \geq 1 \\
 & \text{cycle 4, delay } (g,h)\text{:}\ \ \mathtt{f}_{(g,h)} \geq 1
\end{array}
$$

Table of contents

Evaluation

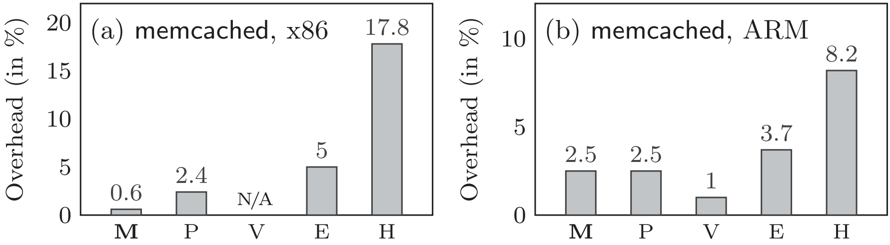
- Measure how well did we do?

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,**P**ensieve,**V**isual Studio, after **E**ach access, after **H**eap 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?