Optimal Reads-From Consistency Checking for C11-Style Memory Models

Hünkar Can Tunç Parosh Aziz Abdulla Soham Chakraborty

Shankaranarayanan Krishna Umang Mathur Andreas Pavlogiannis

AARHUS UNIVERSITY
UPPSALA UNIVERSITET
TU Delft
National University of Singapore

Overview

Scenario I

$$x := 0, y := 0$$

Thread 1	**Thread 2**
$x := 1$	$y := 1$
$a := y$	$b := x$

Overview

Scenario I

$$x := 0, y := 0$$

Thread 1	**Thread 2**
$x := 1$	$y := 1$
$a := y$	$b := x$

Under sequential consistency:

- Is *Scenario I* possible?

  $\hookrightarrow a = b = 0$

Overview

Scenario I

$x := 0, y := 0$

Thread 1

$x := 1$

$a := y$

Thread 2

$y := 1$

$b := x$

Under sequential consistency:

- Is *Scenario I* possible?

  $\hookrightarrow$ $a = b = 0$ ✗

Overview

Scenario I

$$x := 0, y := 0$$

Thread 1	**Thread 2**
$x := 1$	$y := 1$
$a := y$	$b := x$

Scenario II

$$x := 0, y := 0$$

Thread 1	**Thread 2**
$x := 1$	$y := 1$
$a := y$	$b := x$

Under sequential consistency:

- Is *Scenario I* possible?

  $\hookrightarrow a = b = 0$ ✗

- Is *Scenario II* possible?

  $\hookrightarrow a = b = 1$

Overview

Scenario I

$x := 0, y := 0$

Thread 1	**Thread 2**
$x := 1$	$y := 1$
$a := y$	$b := x$

Scenario II

$x := 0, y := 0$

Thread 1	**Thread 2**
$x := 1$	$y := 1$
$a := y$	$b := x$

Under sequential consistency:

- Is *Scenario I* possible?

  $\hookrightarrow\ a = b = 0$ ✗

- Is *Scenario II* possible?

  $\hookrightarrow\ a = b = 1$ ✓

Overview

Reads-From consistency checking:

Overview

Reads-From consistency checking:

- **Input:** Partial execution X in a memory model $\mathcal{M}$
- **Task:** Check if X can be consistent in $\mathcal{M}$

Overview

Reads-From consistency checking:

- **Input:** Partial execution X in a memory model $\mathcal{M}$
- **Task:** Check if X can be consistent in $\mathcal{M}$
- Practical applications:

  $\hookrightarrow$ Model checking, testing

Overview

Reads-From consistency checking:

- **Input:** Partial execution X in a memory model $\mathcal{M}$
- **Task:** Check if X can be consistent in $\mathcal{M}$
- Practical applications:

  $\hookrightarrow$ Model checking, testing
- Well understood for traditional memory models

  $\hookrightarrow$ Sequential consistency, x86-TSO

Overview

Reads-From consistency checking:

- **Input:** Partial execution X in a memory model $\mathcal{M}$
- **Task:** Check if X can be consistent in $\mathcal{M}$
- Practical applications:
  $\hookrightarrow$ Model checking, testing
- Well understood for traditional memory models
  $\hookrightarrow$ Sequential consistency, x86-TSO
- Little is known about the variants of C11 memory model

Overview

Reads-From consistency checking:

- **Input:** Partial execution X in a memory model $\mathcal{M}$
- **Task:** Check if X can be consistent in $\mathcal{M}$
- Practical applications:
  $\hookrightarrow$ Model checking, testing
- Well understood for traditional memory models
  $\hookrightarrow$ Sequential consistency, x86-TSO
- Little is known about the variants of C11 memory model
  $\hookrightarrow$ **This work fills this gap!**

Contributions

- We study reads-from consistency checking in various variants of C11
- **Main results:**
  - Efficient algorithms

    $\hookrightarrow$ Optimal or nearly-optimal
  - Complexity characterization

    $\hookrightarrow$ Fine-grained optimality or $\mathcal{NP}$-hardness results
  - Empirical evaluation

    $\hookrightarrow$ Shows the impact of new algorithms in practice

C11 Memory Model

- Introduced by the ISO C/C++ 2011 standards

C11 Memory Model

- Introduced by the ISO C/C++ 2011 standards
- Support for low-level atomic operations
  - ↪ Load, Store, RMW
  - ↪ Used for communication between threads

C11 Memory Model

- Introduced by the ISO C/C++ 2011 standards
- Support for low-level atomic operations
  - ↪ Load, Store, RMW
  - ↪ Used for communication between threads
- Memory accesses levels:
  - ↪ Synchronization guarantees
  - ↪ Implementation cost

```
atomic<int> x (0);
x.store(1, memory_order_relaxed);
x.load(memory_order_acquire);
```

Sequentially consistent

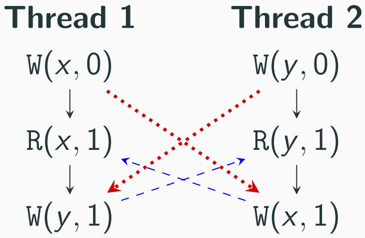
Release Acquire

Relaxed

C11 Memory Model

- Semantics of a program is defined as a set of consistent executions
- Each execution is a graph
  - ↪ Nodes are instructions in the program
  - ↪ Edges represent certain relations among the instructions

Thread 1	**Thread 2**
$x := 0;$	$y := 0;$
$a := x;$	$b := y;$
$y := 1;$	$x := 1;$



C11 Memory Model

- Standard relations:
  $\hookrightarrow$ Program order (po): Precedence among the same thread events

Thread 1	**Thread 2**
$x := 0;$	$y := 0;$
$a := x;$	$b := y;$
$y := 1;$	$x := 1;$

$$\begin{array}{cc}
\textbf{Thread 1} & \textbf{Thread 2} \\
\mathtt{W}(x,0) & \mathtt{W}(y,0) \\
\text{po}\downarrow & \text{po}\downarrow \\
\mathtt{R}(x,?) & \mathtt{R}(y,?) \\
\text{po}\downarrow & \text{po}\downarrow \\
\mathtt{W}(y,1) & \mathtt{W}(x,1)
\end{array}$$

C11 Memory Model

- Standard relations:
  - ↪ Program order (po): Precedence among the same thread events
  - ↪ Reads-from (rf): Relates the writes to the loads which read their value

Thread 1	**Thread 2**
$x := 0;$	$y := 0;$
$a := x;$	$b := y;$
$y := 1;$	$x := 1;$

Thread 1	**Thread 2**

$$
\begin{tikzcd}
\mathtt{W}(x,0) \arrow[d] & & \mathtt{W}(y,0) \arrow[d] \\
\mathtt{R}(x,1) \arrow[d] & & \mathtt{R}(y,1) \arrow[d] \\
\mathtt{W}(y,1) \arrow[urr, dashed, "rf"'] & & \mathtt{W}(x,1) \arrow[ull, dashed, "rf"]
\end{tikzcd}
$$

C11 Memory Model

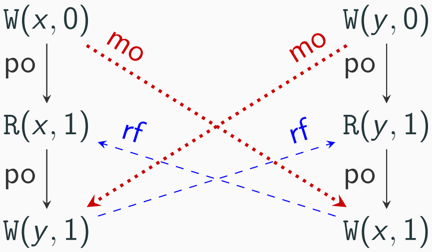
- Standard relations:
  - $\hookrightarrow$ Program order (po): Precedence among the same thread events
  - $\hookrightarrow$ Reads-from (rf): Relates the writes to the loads which read their value
  - $\hookrightarrow$ Modification order (mo): A total order of the writes on a given location

Thread 1	**Thread 2**
$x := 0;$	$y := 0;$
$a := x;$	$b := y;$
$y := 1;$	$x := 1;$

$$
\begin{tikzcd}
\textbf{Thread 1} & \textbf{Thread 2} \\
\mathtt{W}(x, 0) \arrow[d] \arrow[ddr, dotted, "mo"] & \mathtt{W}(y, 0) \arrow[d] \arrow[ddl, dotted, "mo"'] \\
\mathtt{R}(x, ?) \arrow[d] & \mathtt{R}(y, ?) \arrow[d] \\
\mathtt{W}(y, 1) & \mathtt{W}(x, 1)
\end{tikzcd}
$$

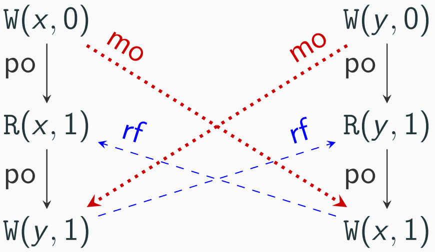
C11-Style Memory Models

- A memory model restricts which executions are consistent



C11-Style Memory Models

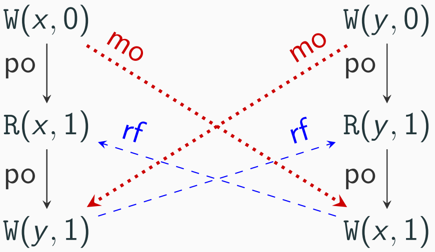
- A memory model restricts which executions are consistent



Memory model $\mathcal{M}1$: acyclic(po $\cup$ mo)

C11-Style Memory Models

- A memory model restricts which executions are consistent

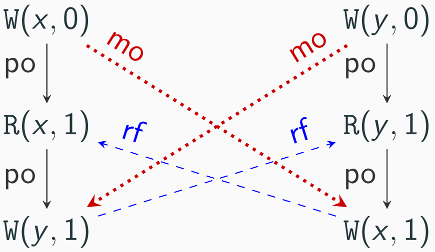


$\rightarrow$ **Consistent in $\mathcal{M}1$ ✓**

Memory model $\mathcal{M}1$: acyclic(po $\cup$ mo)

C11-Style Memory Models

- A memory model restricts which executions are consistent



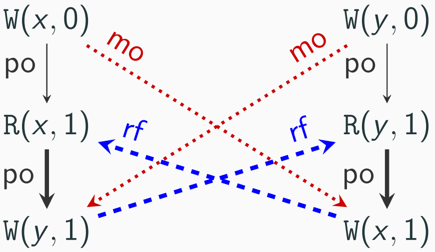
$\rightarrow$ **Consistent in $\mathcal{M}1$ ✓**

Memory model $\mathcal{M}1$: $\mathrm{acyclic}(\mathrm{po} \cup \mathrm{mo})$

Memory model $\mathcal{M}2$: $\mathrm{acyclic}(\mathrm{po} \cup \mathrm{mo}) \wedge \mathrm{acyclic}(\mathrm{po} \cup \mathrm{rf})$

C11-Style Memory Models

- A memory model restricts which executions are consistent



$\rightarrow$ **Consistent in $\mathcal{M}1$ ✓**

$\rightarrow$ **Not consistent in $\mathcal{M}2$ ✗**

Memory model $\mathcal{M}1$: $\text{acyclic}(\text{po} \cup \text{mo})$

Memory model $\mathcal{M}2$: $\text{acyclic}(\text{po} \cup \text{mo}) \wedge \text{acyclic}(\text{po} \cup \text{rf})$

Reads-From Consistency Checking

- **Input:** A partial execution X in a memory model $\mathcal{M}$
  - $\hookrightarrow$ X contains po and rf
  - $\hookrightarrow$ X lacks mo

Reads-From Consistency Checking

- **Input:** A partial execution X in a memory model $\mathcal{M}$
  - $\hookrightarrow$ X contains po and rf
  - $\hookrightarrow$ X lacks mo
- **Task:** Check if X can be extended to a complete execution consistent in $\mathcal{M}$
  - $\hookrightarrow$ Find an mo that turns X consistent

RC20 Memory Model

- Captures a rich fragment of C11
  - ↪ Contains `Release`, `Acquire` and `Relaxed` accesses
  - ↪ Lacks `Sequentially Consistent` accesses

RC20 Memory Model

- Captures a rich fragment of C11
  - $\hookrightarrow$ Contains `Release`, `Acquire` and `Relaxed` accesses
  - $\hookrightarrow$ Lacks `Sequentially Consistent` accesses
- Reads-from consistency checking is a bottleneck
- **Previous works:** $O(n^3 \cdot k)$, $O(n^2 \cdot k)$
  - $\hookrightarrow$ For n events and k threads

RC20 Memory Model

- Captures a rich fragment of C11
  - $\hookrightarrow$ Contains `Release`, `Acquire` and `Relaxed` accesses
  - $\hookrightarrow$ Lacks `Sequentially Consistent` accesses
- Reads-from consistency checking is a bottleneck
- **Previous works:** $O(n^3 \cdot k)$, $O(n^2 \cdot k)$
  - $\hookrightarrow$ For n events and k threads
- **Our result:** $O(n \cdot k)$
  - $\hookrightarrow$ Key idea: *minimal coherence* witness relation

Witness Relation

Witness relation

- ↪ Serves as a witness for consistency
- ↪ Construct a partially ordered mo
    - ↪ Include *necessary* orderings enforced by the memory model
    - ↪ It should be extendable to a total mo

Witness Relation

Witness relation

$\hookrightarrow$ Serves as a witness for consistency

$\hookrightarrow$ Construct a partially ordered mo

- $\hookrightarrow$ Include *necessary* orderings enforced by the memory model
- $\hookrightarrow$ It should be extendable to a total mo

Recall $\rightarrow$

Reads-from consistency checking:

$\hookrightarrow$ Task is to find an mo

$\hookrightarrow$ mo is a total order on the same location writes

Witness Relation

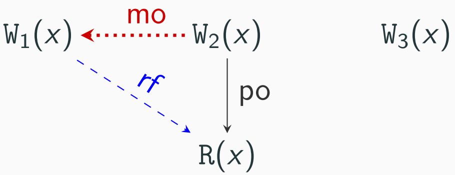
$$
\begin{tikzcd}
\mathtt{W}_1(x) \arrow[dr, dashed, "rf"] & \mathtt{W}_2(x) \arrow[d, "\mathrm{po}"] & \mathtt{W}_3(x) \\
& \mathtt{R}(x) &
\end{tikzcd}
$$

Witness Relation

$$
\begin{tikzcd}
\mathtt{W}_1(x) \arrow[dr, dashed, "rf"] & \mathtt{W}_2(x) \arrow[l, dotted, "mo"'] \arrow[d, "po"] & \mathtt{W}_3(x) \\
& \mathtt{R}(x) &
\end{tikzcd}
$$

- $\mathtt{W}_2(x)$ must be mo ordered before $\mathtt{W}_1(x)$

Witness Relation



- $W_2(x)$ must be mo ordered before $W_1(x)$
- $W_3(x)$ is not relevant
  - $\hookrightarrow$ It can be left unordered

Witness Relation

$$
\begin{tikzcd}
\mathtt{W}_1(x) \arrow[dr, dashed, "rf"] & \mathtt{W}_2(x) \arrow[l, dotted, "\mathsf{mo}"'] \arrow[d, "\mathsf{po}"] & \mathtt{W}_3(x) \\
& \mathtt{R}(x) &
\end{tikzcd}
$$

- $\mathtt{W}_2(x)$ must be mo ordered before $\mathtt{W}_1(x)$
- $\mathtt{W}_3(x)$ is not relevant
  - $\hookrightarrow$ It can be left unordered
- Witness should always be extendable to a total mo
  - $\hookrightarrow$ $\mathtt{W}_3(x) \xrightarrow{\mathsf{mo}} \mathtt{W}_2(x)$
  - $\hookrightarrow$ $\mathtt{W}_1(x) \xrightarrow{\mathsf{mo}} \mathtt{W}_3(x)$

Minimal Coherence for RC20

Minimal coherence

$\hookrightarrow$ Serves as a witness for consistency

$\hookrightarrow$ Weaker than prior witness relations

$\hookrightarrow$ Allows efficient consistency checking algorithm for RC20

Minimal Coherence for RC20

Minimal coherence

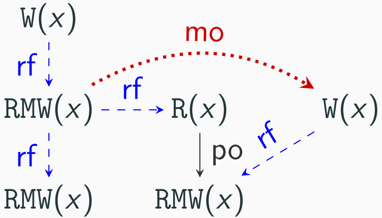
$$
\begin{tikzcd}
\mathtt{W}(x) \arrow[d, dashed, "\mathsf{rf}"'] & & \\
\mathtt{RMW}(x) \arrow[r, dashed, "\mathsf{rf}"] \arrow[d, dashed, "\mathsf{rf}"'] & \mathtt{R}(x) \arrow[d, "\mathsf{po}"] & \mathtt{W}(x) \arrow[dl, dashed, "\mathsf{rf}"'] \\
\mathtt{RMW}(x) & \mathtt{RMW}(x) &
\end{tikzcd}
$$

Prior witness relations

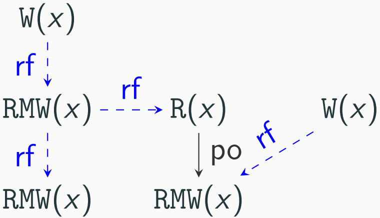
$$
\begin{tikzcd}
\mathtt{W}(x) \arrow[d, dashed, "\mathsf{rf}"'] & & \\
\mathtt{RMW}(x) \arrow[r, dashed, "\mathsf{rf}"] \arrow[d, dashed, "\mathsf{rf}"'] & \mathtt{R}(x) \arrow[d, "\mathsf{po}"] & \mathtt{W}(x) \arrow[dl, dashed, "\mathsf{rf}"'] \\
\mathtt{RMW}(x) & \mathtt{RMW}(x) &
\end{tikzcd}
$$

Minimal Coherence for RC20

Minimal coherence

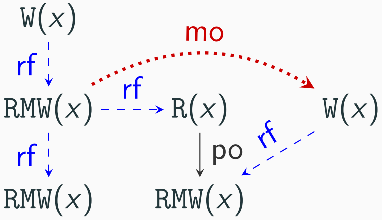


Prior witness relations

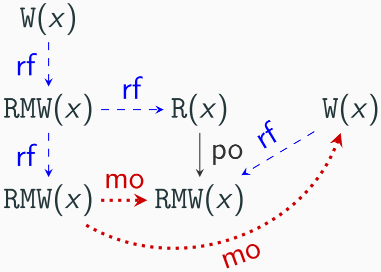


Minimal Coherence for RC20

Minimal coherence

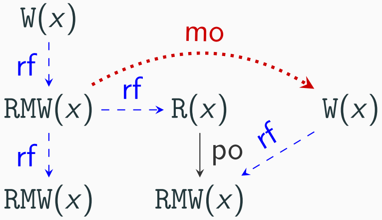


Prior witness relations

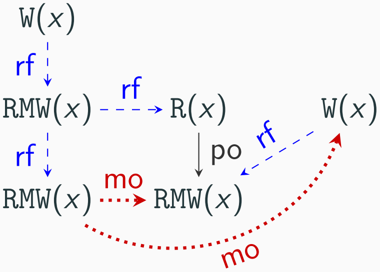


Minimal Coherence for RC20

Minimal coherence



Prior witness relations



- Minimal coherence is weaker!

  ↪ More efficient to compute

Experimental Results

- Focused on the RC20/Release-Acquire (RA) fragments
  - $\hookrightarrow$ RA is a fragment of RC20
- Performed an evaluation in two scenarios
  - $\hookrightarrow$ Model checking
  - $\hookrightarrow$ Testing
- Modified only the consistency checking components

Experimental Results - Model Checking

- Implemented minimal coherence inside GenMC[1]
- Compared with the original GenMC
  - ↪ 25 standard benchmarks
  - ↪ 2 hour timeout

	GenMC	**Our Algorithm**
Average time per execution	14.5 sec	0.26 sec
Total number of executions	356K	4.6M

[1] Michalis Kokologiannakis, Viktor Vafeiadis. GenMC: A Model Checker for Weak Memory Models. CAV'21

Experimental Results - Testing

- Implemented minimal coherence inside C11Tester[1]
  - ↪ Online version
  - ↪ $O(n \cdot k)$ bound does not apply
- Compared with the original C11Tester
  - ↪ 32 standard benchmarks

	C11Tester	**Our Algorithm**
Total analysis time	286 sec	170 sec

[1]Weiyu Luo, Brian Demsky. C11Tester: a race detector for C/C++ atomics. ASPLOS'21

Summary of Established Bounds

- **NP-hard**: $O(k \cdot n^{k+1})$
  $\hookrightarrow$ Strong Release-Acquire (SRA)

$n \rightarrow$ number of events

$k \rightarrow$ number of threads

Summary of Established Bounds

- **NP-hard**: $O(k \cdot n^{k+1})$
  $\hookrightarrow$ Strong Release-Acquire (SRA)
- **Super-linear**: $O(n \cdot k)$
  $\hookrightarrow$ RC20
  $\hookrightarrow$ Weak Release-Acquire (WRA)
  $\hookrightarrow$ RMW-free SRA

$n \rightarrow$ number of events

$k \rightarrow$ number of threads

Summary of Established Bounds

- **NP-hard**: $O(k \cdot n^{k+1})$
  - $\hookrightarrow$ Strong Release-Acquire (SRA)
- **Super-linear**: $O(n \cdot k)$
  - $\hookrightarrow$ RC20
  - $\hookrightarrow$ Weak Release-Acquire (WRA)
  - $\hookrightarrow$ RMW-free SRA
- **Linear**: $O(n)$
  - $\hookrightarrow$ Relaxed

$n \rightarrow$ number of events

$k \rightarrow$ number of threads

Summary of Established Bounds

- **NP-hard**: $O(k \cdot n^{k+1})$
  $\hookrightarrow$ Strong Release-Acquire (SRA)
- **Super-linear**: $O(n \cdot k)$
  $\hookrightarrow$ RC20
  $\hookrightarrow$ Weak Release-Acquire (WRA)
  $\hookrightarrow$ RMW-free SRA
- **Linear**: $O(n)$
  $\hookrightarrow$ Relaxed
- **Super-linear lower bound:**
  $\hookrightarrow$ RMW-free RA, SRA, WRA
  $\hookrightarrow$ Improving $O(n \cdot k)$ bounds would be non-trivial

$n \rightarrow$ number of events

$k \rightarrow$ number of threads

Conclusion

- Addressed the reads-from consistency checking problem in variants of C11

Conclusion

- Addressed the reads-from consistency checking problem in variants of C11
- Collection of optimal or nearly-optimal algorithms for different variants

Conclusion

- Addressed the reads-from consistency checking problem in variants of C11
- Collection of optimal or nearly-optimal algorithms for different variants
- Established fine-grained complexity results

Conclusion

- Addressed the reads-from consistency checking problem in variants of C11
- Collection of optimal or nearly-optimal algorithms for different variants
- Established fine-grained complexity results
- Experimental evaluation confirms the impact of the new algorithms

Conclusion

- Addressed the reads-from consistency checking problem in variants of C11
- Collection of optimal or nearly-optimal algorithms for different variants
- Established fine-grained complexity results
- Experimental evaluation confirms the impact of the new algorithms

Thank you!