Sequential Reasoning for Optimizing Compilers Under Weak Memory Concurrency

Minki Cho, Sung Hwan Lee, Dongjae Lee, Chung Kil Hur, Ori Lahav

Presented by
Akshay Gopalakrishnan

April 10, 2024

Introduction

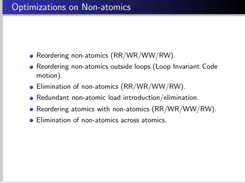
- Performing/designing thread-local optimizations in a concurrent context would require understanding the underlying memory consistency model.
- Often, these models are complex, as they are weak consistency based.
- Thus, it becomes difficult to design optimizations which are safe.
- Most optimizations that are performed on concurrent programs mainly involve reordering or eliminating non-atomics.
- Is there perhaps, then a simpler semantics that an optimization designer can rely on?

Main Idea

- This paper proposes a Sequential Model (SEQ) to design compiler optimizations.
- The optimizations addressed are solely regarding non-atomics, or reordering non-atomics across atomic parts of code.
- SEQ is shown to preserve contextual refinement for non-atomic optimizations in the Promising Semantics (PS).
- Contextual refinement in PS makes sure that optimization developers can solely on SEQ to design optimizations.

Optimizations on Non-atomics

- Reordering non-atomics (RR/WR/WW/RW).
- Reordering non-atomics outside loops (Loop Invariant Code motion).
- Elimination of non-atomics (RR/WR/WW/RW).
- Redundant non-atomic load introduction/elimination.
- Reordering atomics with non-atomics (RR/WR/WW/RW).
- Elimination of non-atomics across atomics.


Optimizations on Non-atomics

- Reordering non-atomics (RR/WR/WW/RW).
- Reordering non-atomics outside loops (Loop Invariant Code motion).
- Elimination of non-atomics (RR/WR/WW/RW).
- Redundant non-atomic load introduction/elimination.
- Reordering atomics with non-atomics (RR/WR/WW/RW).
- Elimination of non-atomics across atomics.

The non-adjacent optimization involving elimination of non-atomics across atomics does not seem to be required in my opinion. This is because the proof for that can be justified by decomposing the optimizations into the above elementary adjacent components.

To add, Write introduction is not allowed by the model. This is a good design choice to preserve security guarantees, however it still limits from introducing redundant writes within dead-code blocks, which may be used by optimizing compilers to easily perform partial redundancy elimination.

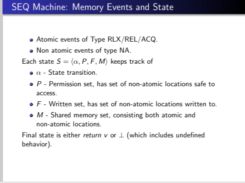
SEQ Machine: Memory Events and State

- Atomic events of Type RLX/REL/ACQ.
- Non atomic events of type NA.

Each state $S = \langle \alpha, P, F, M \rangle$ keeps track of

- α - State transition.
- P - Permission set, has set of non-atomic locations safe to access.
- F - Written set, has set of non-atomic locations written to.
- M - Shared memory set, consisting both atomic and non-atomic locations.

Final state is either *return v* or $\perp$ (which includes undefined behavior).



SEQ Machine: Memory Events and State

- Atomic events of Type RLX/REL/ACQ.
- Non atomic events of type NA.

Each state $S = \langle \alpha, P, F, M \rangle$ keeps track of

- α - State transition.
- P - Permission set, has set of non-atomic locations safe to access.
- F - Written set, has set of non-atomic locations written to.
- M - Shared memory set, consisting both atomic and non-atomic locations.

Final state is either *return* v or $\perp$ (which includes undefined behavior).

It is unclear whether P at the beginning of the execution includes all possible non-atomic locations (is it $\top$) or it is empty (is it $\perp$). My understanding is that it should be $\perp$, and that acquire semantics would start including in the set locations we are safe to read/write to, whereas release semantics would do the reverse. The transitions seem to make sense, the non-deterministic change (inclusion in acq and exclusion in rel) is meant to reflect any random context (other threads) that would be running concurrently.

SEQ Machine 1: State Transitions

(SILENT)
$$\frac{\sigma \to \sigma'}{\langle \sigma, P, F, M\rangle \to \langle \sigma', P, F, M\rangle}$$

(CHOICE/RELAXED)
$$\frac{\sigma \xrightarrow{e} \sigma' \qquad e \in \{\mathtt{choose}(v), \mathtt{R}^{\mathtt{rlx}}(x, v), \mathtt{W}^{\mathtt{rlx}}(x, v)\}}{\langle \sigma, P, F, M\rangle \xrightarrow{e} \langle \sigma', P, F, M\rangle}$$

(NA-READ)
$$\frac{\sigma \xrightarrow{\mathtt{R}^{\mathtt{na}}(x, v)} \sigma' \qquad x \in P \qquad v = M(x)}{\langle \sigma, P, F, M\rangle \to \langle \sigma', P, F, M\rangle}$$

(NA-WRITE)
$$\frac{\sigma \xrightarrow{\mathtt{W}^{\mathtt{na}}(x, v)} \sigma' \qquad x \in P \qquad F' = F \cup \{x\} \qquad M' = M[x \mapsto v]}{\langle \sigma, P, F, M\rangle \to \langle \sigma', P, F', M'\rangle}$$

(RACY-NA-READ)
$$\frac{\sigma \xrightarrow{\mathtt{R}^{\mathtt{na}}(x, \mathtt{undef})} \sigma' \qquad x \notin P}{\langle \sigma, P, F, M\rangle \to \langle \sigma', P, F, M\rangle}$$

(RACY-NA-WRITE)
$$\frac{\sigma \xrightarrow{\mathtt{W}^{\mathtt{na}}(x, \_)} \_ \qquad x \notin P}{\langle \sigma, P, F, M\rangle \to \langle \bot, P, F, M\rangle}$$

(ACQ-READ)
$$\frac{\sigma \xrightarrow{\mathtt{R}^{\mathtt{acq}}(x, v)} \sigma' \qquad P \subseteq P' \qquad dom(V) = P' \setminus P \qquad M' = \lambda x. \begin{cases} V(x) & x \in P' \setminus P \\ M(x) & \text{otherwise} \end{cases}}{\langle \sigma, P, F, M\rangle \xrightarrow{\mathtt{R}^{\mathtt{acq}}(x, v, P, P', F, V)} \langle \sigma', P', F, M'\rangle}$$

(REL-WRITE)
$$\frac{\sigma \xrightarrow{\mathtt{W}^{\mathtt{rel}}(x, v)} \sigma' \qquad P' \subseteq P \qquad V = M|_P}{\langle \sigma, P, F, M\rangle \xrightarrow{\mathtt{W}^{\mathtt{rel}}(x, v, P, P', F, V)} \langle \sigma', P', \emptyset, M\rangle}$$

Figure 1. Transitions of SEQ

Towards Soundness of Optimizations

A Behavior is defined as $\langle tr, r\rangle$ where

- tr is the sequence of labels denoting an in which a trace of events were executed.
- r is either
  - $trm(v, F, M)$ denoting return value v, updated writes F and memory state M.
  - $\perp$ denoting erroneous termination, which includes Undefined behavior.

Refinement of Transition Steps

1. Transition labels:

$$\frac{}{e \sqsubseteq e} \qquad \frac{v_{\text{tgt}} \sqsubseteq v_{\text{src}}}{\mathtt{W}^{\mathtt{rlx}}(x, v_{\text{tgt}}) \sqsubseteq \mathtt{W}^{\mathtt{rlx}}(x, v_{\text{src}})} \qquad \frac{F_{\text{tgt}} \subseteq F_{\text{src}}}{\mathtt{R}^{\mathtt{acq}}(x, v, P, P', F_{\text{tgt}}, V) \sqsubseteq \mathtt{R}^{\mathtt{acq}}(x, v, P, P', F_{\text{src}}, V)}$$

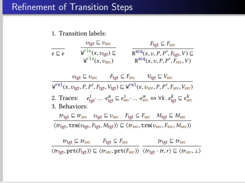
$$\frac{v_{\text{tgt}} \sqsubseteq v_{\text{src}} \qquad F_{\text{tgt}} \subseteq F_{\text{src}} \qquad V_{\text{tgt}} \sqsubseteq V_{\text{src}}}{\mathtt{W}^{\mathtt{rel}}(x, v_{\text{tgt}}, P, P', F_{\text{tgt}}, V_{\text{tgt}}) \sqsubseteq \mathtt{W}^{\mathtt{rel}}(x, v_{\text{src}}, P, P', F_{\text{src}}, V_{\text{src}})}$$

2. Traces: $e^1_{\text{tgt}} \cdot \ldots \cdot e^n_{\text{tgt}} \sqsubseteq e^1_{\text{src}} \cdot \ldots \cdot e^n_{\text{src}} \Leftrightarrow \forall k.\ e^k_{\text{tgt}} \sqsubseteq e^k_{\text{src}}$

3. Behaviors:

$$\frac{tr_{\text{tgt}} \sqsubseteq tr_{\text{src}} \qquad v_{\text{tgt}} \sqsubseteq v_{\text{src}} \qquad F_{\text{tgt}} \subseteq F_{\text{src}} \qquad M_{\text{tgt}} \sqsubseteq M_{\text{src}}}{\langle tr_{\text{tgt}}, \mathtt{trm}(v_{\text{tgt}}, F_{\text{tgt}}, M_{\text{tgt}})\rangle \sqsubseteq \langle tr_{\text{src}}, \mathtt{trm}(v_{\text{src}}, F_{\text{src}}, M_{\text{src}})\rangle}$$

$$\frac{tr_{\text{tgt}} \sqsubseteq tr_{\text{src}} \qquad F_{\text{tgt}} \subseteq F_{\text{src}}}{\langle tr_{\text{tgt}}, \mathtt{prt}(F_{\text{tgt}})\rangle \sqsubseteq \langle tr_{\text{src}}, \mathtt{prt}(F_{\text{src}})\rangle} \qquad \frac{tr_{\text{tgt}} \sqsubseteq tr_{\text{src}}}{\langle tr_{\text{tgt}} \cdot tr, r\rangle \sqsubseteq \langle tr_{\text{src}}, \bot\rangle}$$



Refinement of Transition Steps

1. Transition labels:

$$\frac{}{e \sqsubseteq e} \qquad \frac{v_{\mathrm{tgt}} \sqsubseteq v_{\mathrm{src}}}{\mathtt{W}^{\mathtt{rlx}}(x, v_{\mathrm{tgt}}) \sqsubseteq \mathtt{W}^{\mathtt{rlx}}(x, v_{\mathrm{src}})} \qquad \frac{F_{\mathrm{tgt}} \subseteq F_{\mathrm{src}}}{\mathtt{R}^{\mathtt{acq}}(x, v, P, P', F_{\mathrm{tgt}}, V) \sqsubseteq \mathtt{R}^{\mathtt{acq}}(x, v, P, P', F_{\mathrm{src}}, V)}$$

$$\frac{v_{\mathrm{tgt}} \sqsubseteq v_{\mathrm{src}} \qquad F_{\mathrm{tgt}} \subseteq F_{\mathrm{src}} \qquad V_{\mathrm{tgt}} \sqsubseteq V_{\mathrm{src}}}{\mathtt{W}^{\mathtt{rel}}(x, v_{\mathrm{tgt}}, P, P', F_{\mathrm{tgt}}, V_{\mathrm{tgt}}) \sqsubseteq \mathtt{W}^{\mathtt{rel}}(x, v_{\mathrm{src}}, P, P', F_{\mathrm{src}}, V_{\mathrm{src}})}$$

2. Traces: $e^1_{\mathrm{tgt}} \cdot \ldots \cdot e^n_{\mathrm{tgt}} \sqsubseteq e^1_{\mathrm{src}} \cdot \ldots \cdot e^n_{\mathrm{src}} \Leftrightarrow \forall k.\ e^k_{\mathrm{tgt}} \sqsubseteq e^k_{\mathrm{src}}$
3. Behaviors:

$$\frac{tr_{\mathrm{tgt}} \sqsubseteq tr_{\mathrm{src}} \quad v_{\mathrm{tgt}} \sqsubseteq v_{\mathrm{src}} \quad F_{\mathrm{tgt}} \subseteq F_{\mathrm{src}} \quad M_{\mathrm{tgt}} \sqsubseteq M_{\mathrm{src}}}{\langle tr_{\mathrm{tgt}}, \mathtt{trm}(v_{\mathrm{tgt}}, F_{\mathrm{tgt}}, M_{\mathrm{tgt}}) \rangle \sqsubseteq \langle tr_{\mathrm{src}}, \mathtt{trm}(v_{\mathrm{src}}, F_{\mathrm{src}}, M_{\mathrm{src}}) \rangle}$$

$$\frac{tr_{\mathrm{tgt}} \sqsubseteq tr_{\mathrm{src}} \qquad F_{\mathrm{tgt}} \subseteq F_{\mathrm{src}}}{\langle tr_{\mathrm{tgt}}, \mathtt{prt}(F_{\mathrm{tgt}}) \rangle \sqsubseteq \langle tr_{\mathrm{src}}, \mathtt{prt}(F_{\mathrm{src}}) \rangle} \quad \frac{tr_{\mathrm{tgt}} \sqsubseteq tr_{\mathrm{src}}}{\langle tr_{\mathrm{tgt}} \cdot tr, r \rangle \sqsubseteq \langle tr_{\mathrm{src}}, \bot \rangle}$$

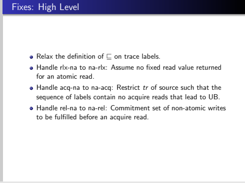
What baffles me is the last transition, which says the normal termination is a refinement of erroenous termination. I guess this means it is okay to make something that invokes undefined behavior to one that does not. In the context of data races however, this would clearly be incorrect, since removing races, even if desirable is not the original behavior of the program. The compiler is allowed to do anything yes, but I guess from a debugging standpoint this would not be ideal. Perhaps mail one of the authors about this.

Problem with Simple SEQ

- Reordering of non-atomics across atomics in certain cases are unsound.
- Most of these cases reduce to *Roach Motel Reordering*.
- Reason is the final set of labels are different, keeping the termination state as $\perp$.
- Example: $a = y^{rlx}; b = x^{na} \mapsto b = x^{na}; a = y^{rlx}$.

Fixes: High Level

- Relax the definition of $\sqsubseteq$ on trace labels.
- Handle rlx-na to na-rlx: Assume no fixed read value returned for an atomic read.
- Handle acq-na to na-acq: Restrict *tr* of source such that the sequence of labels contain no acquire reads that lead to UB.
- Handle rel-na to na-rel: Commitment set of non-atomic writes to be fulfilled before an acquire read.



Fixes: High Level

- Relax the definition of $\sqsubseteq$ on trace labels.
- Handle rlx-na to na-rlx: Assume no fixed read value returned for an atomic read.
- Handle acq-na to na-acq: Restrict *tr* of source such that the sequence of labels contain no acquire reads that lead to UB.
- Handle rel-na to na-rel: Commitment set of non-atomic writes to be fulfilled before an acquire read.

The solution to address the disallowing of non-atomic and atomic reordering is certainly confusing. While some of it is intuitive, it does not give the confidence that it actually would hold in all cases. The crux is to relax the refinement definition to incorporate more reorderings, noting that some information of trace labels are irrelevant.

- The first idea is to note that if you anyways reach UB irrespective of reordering, then you can make do by not ensuring all trace labels are the same.
- To ensure we do not mess this up, we need to ensure that we do not introduce UB by reordering across acquire reads (as hb relation would make sure no data race in some sense). This translates to ensuring the trace labels have no trailing acquire reads.
- Next, to ensure we do not introduce UB again by optimizing, we should not assume the atomic reads have a fixed read value, assuming no fixed value of shared memory.

Towards Adequacy of SEQ

The main claim for SEQ was that a compiler developer can solely rely on sequential semantics (SEQ in this case) to design sound optimizations (that which satisfy contextual refinement).

- Adequacy of SEQ is shown by claiming any optimization (refinement) sound in SEQ is sound in Promising Semantics.
- Adequacy is contingent on program being deterministic, that concrete execution path always leads to the same final outcome.

Assumptions/Limitations

- Do not tackle optimizations only involving atomics, claiming this is rarely done by compilers.
- LICM tackled only involves removing read-events outside the loop, what about writes? Usually writes are not involved or considered as a Loop invariant.
- SEQ thus, cannot be used for atomic optimizations, and thus atomic based optimizations would require understanding PS at its full.
- SEQ relies on the practical implementation and usage of Promising Semantics as a programming language model.
- Paper claims it addresses optimizations without catch-fire semantics, but still uses a notion of UB from LLVM semantics to prove soundness of optimizations.
- Paper's claims on optimizations are additive, with minor warnings on allowing false optimizations. However, it is not yet clear if it allows only what is desired.

Conclusion

- Paper seemed to falsify the rhetoric. The intention being to allow non-atomic based optimizations in the PS.
- This was done by designing a simpler sequential model SEQ, taking advantage of UB and being conservative while proving contextual refinement.
- SEQ was then showed COMPLETE w.r.t. PS for the intended optimizations.
- Promising Semantics was extended by adding non-atomic events (did not discuss here in this presentation).