Extending Intermediate Memory Model with SC accesses.

Technical Report

ANTON PODKOPAEV, NRU-HSE, Russia and MPI-SWS, Germany
ORI LAHAV, Tel Aviv University, Israel
ORESTIS MELKONIAN, MPI-SWS, Germany
VIKTOR VAFEIADIS, MPI-SWS, Germany

IMM, an intermediate memory model, abstracts x86-TSO, POWER, ARMv7, ARMv8, and RISC-V memory models (MM). Through this model, Podkopaev et al. proved correctness of compilation from programming language MMs (the promising semantics of Kang et al., the C/C++11 MM and RC11 MM of Lahav et al.) to the abstracted hardware MMs. IMM lacks SC accesses, and because of that the compilation correctness results for the C/C++11 and RC11 MMs cover only subsets of the models without SC accesses. Also, the absence of SC accesses makes IMM unsuitable for proving correctness of compilation from OCaml (Dolan et al.) and JavaScript MMs since the models have rather strong accesses, which are close to SC accesses in terms of the C/C++11 MM. We propose an extension of IMM with SC accesses, called IMM_{SC}, and update correctness of compilation proofs from the programming language memory models to IMM and from IMM to the hardware memory models. The extension follows the fix of SC access semantics proposed in Lahav et al. The Coq mechanization of IMM and the related compilation correctness proofs are correspondingly extended.

Additional Key Words and Phrases: weak memory consistency, declarative semantics, IMM, C11, RC11

Disclaimer: In its current form, this document is not supposed to be self-contained but rather to be an extension of [Podkopaev et al. 2019]. That is, a number of definitions used in this text are not presented here, and a reader is expected to be familiar with [Podkopaev et al. 2019].

1 IMM_{SC}: IMM EXTENDED WITH SC ACCESSES

In this section, we start by introducing IMM_{SC}-consistency predicate as it is and then discuss why it is defined in that way.

1.1 IMM_{SC}-consistency predicate

To define the consistency predicate of IMM_{SC}, we need to extend a notion of IMM execution graphs—they should be allowed to have read and write accesses with sc modifiers. Correspondingly, the source language, for which programs IMM execution graphs are constructed, is also revised to allow SC read and write instructions. Also, we update definitions of relations used in IMM consistency predicate, *i.e.,* release, sw, and bob, by replacing $\mathtt{R}^{\mathtt{acq}}$ and $\mathtt{W}^{\mathtt{rel}}$ in their definitions by $\mathtt{R}^{\sqsupseteq\mathtt{acq}}$ and $\mathtt{W}^{\sqsupseteq\mathtt{rel}}$ respectively since SC accesses are supposed to provide at least the same synchronization guarantees as acquire and release accesses:

$$\mathtt{release} \triangleq ([\mathtt{W}^{\sqsupseteq\mathtt{rel}}] \cup [\mathtt{F}^{\sqsupseteq\mathtt{rel}}] ; \mathsf{po}) ; \mathtt{rs}$$

$$\mathtt{sw} \triangleq \mathtt{release} ; (\mathtt{rfi} \cup \mathsf{po}|_{\mathsf{loc}}^{?} ; \mathtt{rfe}) ; ([\mathtt{R}^{\sqsupseteq\mathtt{acq}}] \cup \mathsf{po} ; [\mathtt{F}^{\sqsupseteq\mathtt{acq}}])$$

$$\mathsf{bob} \triangleq \mathsf{po} ; [\mathtt{W}^{\sqsupseteq\mathtt{rel}}] \cup [\mathtt{R}^{\sqsupseteq\mathtt{acq}}] ; \mathsf{po} \cup \mathsf{po} ; [\mathtt{F}] \cup [\mathtt{F}] ; \mathsf{po} \cup [\mathtt{W}^{\sqsupseteq\mathtt{rel}}] ; \mathsf{po}|_{\mathsf{loc}} ; [\mathtt{W}]$$

Authors' addresses: Anton Podkopaev, NRU-HSE, Russia , MPI-SWS, Saarland Informatics Campus, Germany, podkopaev@mpi-sws.org; Ori Lahav, Tel Aviv University, Israel, orilahav@tau.ac.il; Orestis Melkonian, MPI-SWS, Saarland Informatics Campus, Germany, melkon.or@gmail; Viktor Vafeiadis, MPI-SWS, Saarland Informatics Campus, Germany, viktor@mpi-sws.org.

2019. 2475-1421/2019/1-ART
https://doi.org/

With the modifications, we define IMM-consistency in the same way as in [Podkopaev et al. 2019].

Definition 1.1. G is called IMM-*consistent* if the following hold:

- $\mathit{codom}(G.\mathtt{rf}) = G.\mathsf{R}$. (rf-completeness)
- For every location $x \in \mathrm{Loc}$, $G.\mathtt{co}$ totally orders $G.\mathtt{W}(x)$. (co-totality)
- $G.\mathtt{hb} ; G.\mathtt{eco}^?$ is irreflexive. (coherence)
- $G.\mathtt{rmw} \cap (G.\mathtt{fre} ; G.\mathtt{coe}) = \emptyset$. (atomicity)
- $G.\mathtt{ar}$ is acyclic. (no-thin-air)

$\mathrm{IMM_{SC}}$-consistency has an additional requirement which is the same (up to different definitions of hb, see §2) as the sc axiom in RC11-consistency predicate [Lahav et al. 2017, Definition 1].

Definition 1.2. G is called $\mathrm{IMM_{SC}}$-*consistent* if it is IMM-consistent and

- $G.\mathtt{psc}_{\mathrm{base}} \cup G.\mathtt{psc}_{\mathsf{F}}$ is acyclic. (sc)

Here $\mathtt{psc}_{\mathrm{base}}$ and $\mathtt{psc}_{\mathsf{F}}$ are defined in the following way:

$$\mathtt{scb} \triangleq \mathtt{po} \cup \mathtt{po}|_{\neq \mathtt{loc}} ; \mathtt{hb} ; \mathtt{po}|_{\neq \mathtt{loc}} \cup \mathtt{hb}|_{=\mathtt{loc}} \cup \mathtt{co} \cup \mathtt{fr}$$

$$\mathtt{psc}_{\mathrm{base}} \triangleq ([\mathsf{E}^{\mathtt{sc}}] \cup [\mathsf{F}^{\mathtt{sc}}]; \mathtt{hb}^?) ; \mathtt{scb}; ([\mathsf{E}^{\mathtt{sc}}] \cup \mathtt{hb}^?; [\mathsf{F}^{\mathtt{sc}}])$$

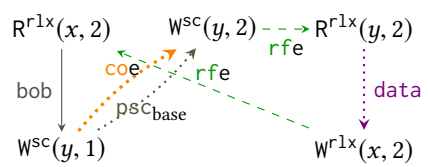
$$\mathtt{psc}_{\mathsf{F}} \triangleq [\mathsf{F}^{\mathtt{sc}}]; (\mathtt{hb} \cup \mathtt{hb}; \mathtt{eco}; \mathtt{hb}); [\mathsf{F}^{\mathtt{sc}}]$$

Note that in definition of IMM-consistency we use the ar relation which includes $\mathtt{psc} \triangleq [\mathsf{F}^{\mathtt{sc}}] ; \mathtt{hb} ; \mathtt{eco} ; \mathtt{hb} ; [\mathsf{F}^{\mathtt{sc}}]$. In [Lahav et al. 2017], psc is defined to be a different thing—a union of $\mathtt{psc}_{\mathrm{base}}$ and $\mathtt{psc}_{\mathsf{F}}$, but in this paper we do not use it.

1.2 Why not to extend ar?

To cover SC accesses in $\mathrm{IMM_{SC}}$-consistency predicate, we put the sc axiom to it. One other option was to try to extend the ar relation with $\mathtt{psc}_{\mathrm{base}}$. However, it leads to the overly strong no-thin-air axiom—the following behaviour becomes forbidden:

$$a := [x]^{\mathtt{rlx}} \;/\!/\; 2 \;\Big\|\; [y]^{\mathtt{sc}} := 1 \qquad [y]^{\mathtt{sc}} := 2 \;\Big\|\; b := [y]^{\mathtt{rlx}} \;/\!/\; 2 \;\Big\|\; [x]^{\mathtt{rlx}} := b$$



It is harmful since the behavior is allowed by POWER assuming both commonly used compilation schemes for SC accesses.

2 FROM C11 AND RC11 TO $\mathrm{IMM_{SC}}$

In this section, we extend the correctness of the mapping from C11 and RC11 models to IMM to cover SC accesses, *i.e.,* establish the correctness of the mapping to $\mathrm{IMM_{SC}}$. This extension is straightforward since $\mathrm{IMM_{SC}}$ has almost the same sc axiom as RC11.

Following Podkopaev et al. [2019], we consider a version of the C11 model of Batty et al. [2011] with fixes from Vafeiadis et al. [2015] and Lahav et al. [2017]. In this paper, we contemplate SC accesses of the model but still not the non-atomic fragment of it since $\mathrm{IMM_{SC}}$, like IMM, does not have direct counterparts for non-atomic accesses. That is, we assume the following definition of C11-consistency.

Definition 2.1. G is called C11-*consistent* if the following hold:

- $\mathit{codom}(G.\mathtt{rf}) = G.\mathsf{R}$.

$$(\!|r := [e]^{\mathtt{rlx}}|\!) \approx \text{“}\mathtt{ldr}\text{”} \qquad (\!|[e_1]^{\mathtt{rlx}} := e_2|\!) \approx \text{“}\mathtt{str}\text{”}$$
$$(\!|r := [e]^{\sqsupseteq \mathtt{acq}}|\!) \approx \text{“}\mathtt{ldar}\text{”} \qquad (\!|[e_1]^{\sqsupseteq \mathtt{rel}} := e_2|\!) \approx \text{“}\mathtt{stlr}\text{”}$$
$$(\!|\mathbf{fence}^{\mathtt{acq}}|\!) \approx \text{“}\mathtt{dmb.ld}\text{”} \qquad (\!|\mathbf{fence}^{\neq \mathtt{acq}}|\!) \approx \text{“}\mathtt{dmb.sy}\text{”}$$
$$(\!|r := \mathbf{FADD}^{o_R, o_W}_{o_{RMW}}(e_1, e_2)|\!) \approx \text{“L:”} +\!\!+ \mathrm{ld}(o_R) +\!\!+ \mathrm{st}(o_W) +\!\!+ \text{“}\mathtt{bc\ L}\text{”} +\!\!+ \mathrm{dmb}(o_{RMW})$$
$$(\!|r := \mathbf{CAS}^{o_R, o_W}_{o_{RMW}}(e, e_R, e_W)|\!) \approx \text{“L:”} +\!\!+ \mathrm{ld}(o_R) +\!\!+ \text{“}\mathtt{cmp;bc\ Le;}\text{”} +\!\!+ \mathrm{st}(o_W) +\!\!+ \text{“}\mathtt{bc\ L;Le:}\text{”} +\!\!+ \mathrm{dmb}(o_{RMW})$$
$$\mathrm{ld}(o_R) \triangleq o_R \sqsupseteq \mathtt{acq}\ ?\ \text{“}\mathtt{ldaxr;}\text{”}\ :\ \text{“}\mathtt{ldxr;}\text{”} \qquad \mathrm{st}(o_W) \triangleq o_W \sqsupseteq \mathtt{rel}\ ?\ \text{“}\mathtt{stlxr.;}\text{”}\ :\ \text{“}\mathtt{stxr.;}\text{”}$$
$$\mathrm{dmb}(o_{RMW}) \triangleq o_{RMW} = \mathtt{strong}\ ?\ \text{“}\mathtt{;dmb.ld}\text{”}\ :\ \text{“”}$$

Fig. 1. Compilation scheme from $\mathrm{IMM_{SC}}$ to ARMv8.

- For every location $x \in \mathsf{Loc}$, $G.\mathtt{co}$ totally orders $G.\mathtt{W}(x)$.
- $G.\mathtt{hb}_{\mathrm{RC11}} ; G.\mathtt{eco}^?$ is irreflexive.
- $G.\mathtt{rmw} \cap (G.\mathtt{fre} ; G.\mathtt{coe}) = \emptyset$.
- $[\mathtt{F}^{\mathtt{sc}}] ; (\mathtt{hb}_{\mathrm{RC11}} \cup \mathtt{hb}_{\mathrm{RC11}} ; \mathtt{eco} ; \mathtt{hb}_{\mathrm{RC11}}) ; [\mathtt{F}^{\mathtt{sc}}]$ is acyclic.
- $G.\mathtt{psc}_{\text{base-RC11}} \cup G.\mathtt{psc}_{\text{F-RC11}}$ is acyclic. (SC-RC11)

Here $\mathtt{psc}_{\text{base-RC11}}$ and $\mathtt{psc}_{\text{F-RC11}}$ are the same as $\mathtt{psc}_{\text{base}}$ and $\mathtt{psc}_{\mathtt{F}}$ from Section 1.1 but defined with $\mathtt{hb}_{\mathrm{RC11}}$ instead of hb (see [Podkopaev et al. 2019, Remark 2]). Since $\mathtt{hb}_{\mathrm{RC11}}$ is weaker (namely, it is smaller) than hb, C11-consistency directly follows from $\mathrm{IMM_{SC}}$-consistency.

The RC11-consistency predicate [Lahav et al. 2017] is defined as a strengthening of Def. 2.1 by additionally requiring acyclicity of $G.\mathtt{po} \cup G.\mathtt{rf}$. Since Podkopaev et al. [2019] proved the correctness of a mapping from RC11 to IMM which places a (control) dependency or a fence between every relaxed read and subsequent relaxed write and SC-RC11 directly follows from SC, the same mapping from RC11 to $\mathrm{IMM_{SC}}$ is also correct (in the presence of SC accesses).

3 FROM $\mathrm{IMM_{SC}}$ TO ARMv8

The intended mapping of IMM to ARMv8 is presented schematically in Fig. 1 and follows [Mapping 2016]. Note that acquire and SC loads are compiled to the same instruction (`ldar`) as well as release and SC stores (`stlr`). In ARM assembly RMWs are represented as pairs of instructions—*exclusive* load (`ldxr`) followed by *exclusive* store (`stxr`), and these instructions are also have their stronger (SC) counterparts—`ldaxr` and `stlxr`.

We use ARMv8 declarative model [Deacon 2017] (see also [Pulte et al. 2018]).[1] Its labels are given by:

- ARM read label: $\mathtt{R}^{o_R}(x, v)$ where $x \in \mathsf{Loc}$, $v \in \mathsf{Val}$, $o_R \in \{\mathtt{rlx}, \mathtt{Q}, \mathtt{A}\}$, and $\mathtt{rlx} \sqsubset \mathtt{Q} \sqsubset \mathtt{A}$.
- ARM write label: $\mathtt{W}^{o_W}(x, v)$ where $x \in \mathsf{Loc}$, $v \in \mathsf{Val}$, $o_W \in \{\mathtt{rlx}, \mathtt{L}\}$, and $\mathtt{rlx} \sqsubset \mathtt{L}$.
- ARM fence label: $\mathtt{F}^{o_F}$ where $o_F \in \{\mathtt{ld}, \mathtt{sy}\}$ and $\mathtt{ld} \sqsubset \mathtt{sy}$.

In turn, ARM's execution graphs are defined as $\mathrm{IMM_{SC}}$'s ones, except for the CAS dependency, casdep, which is not present in ARM executions.

The definition of ARMv8-consistency requires the following derived relations (see [Pulte et al. 2018] for further explanations and details):

$$\mathsf{obs} \triangleq \mathtt{rfe} \cup \mathtt{fre} \cup \mathtt{coe} \qquad (\textit{observed-by})$$
$$\mathsf{dob} \triangleq (\mathtt{addr} \cup \mathtt{data}); \mathtt{rfi}^? \cup (\mathtt{ctrl} \cup \mathtt{data}); [\mathtt{W}]; \mathtt{coi}^? \cup \mathtt{addr}; \mathtt{po}; [\mathtt{W}] \qquad (\textit{dependency-ordered-before})$$
$$\mathsf{aob} \triangleq \mathtt{rmw} \cup [\mathtt{W}^{\mathtt{ex}}]; \mathtt{rfi}; [\mathtt{R}^{\sqsupseteq \mathtt{Q}}] \qquad (\textit{atomic-ordered-before})$$

[1] We only describe the fragment of the model that is needed for mapping of $\mathrm{IMM_{SC}}$, thus excluding `isb` fences.

$$\mathsf{bob} \triangleq \mathsf{po};[\mathsf{F}^{\mathsf{sy}}];\mathsf{po} \cup [\mathsf{R}];\mathsf{po};[\mathsf{F}^{\mathsf{ld}}];\mathsf{po} \cup [\mathsf{R}^{\sqsupseteq \mathsf{Q}}];\mathsf{po} \cup \mathsf{po};[\mathsf{W}^{\mathsf{L}}];\mathsf{coi}^? \cup [\mathsf{W}^{\mathsf{L}}];\mathsf{po};[\mathsf{R}^{\mathsf{A}}] \quad \textit{(barrier-ordered-before)}$$

Definition 3.1. An ARMv8 execution graph G_a is called ARMv8-consistent if the following hold:

- $\mathit{codom}(G_a.\mathtt{rf}) = G_a.\mathsf{R}$.
- For every location $x \in \mathsf{Loc}$, $G_a.\mathtt{co}$ totally orders $G_a.\mathsf{W}(x)$.
- $G_a.\mathsf{po}|_{\mathtt{loc}} \cup G_a.\mathtt{rf} \cup G_a.\mathtt{fr} \cup G_a.\mathtt{co}$ is acyclic. (SC-PER-LOC)
- $G_a.\mathtt{rmw} \cap (G_a.\mathtt{fre}; G_a.\mathtt{coe}) = \emptyset$.
- $G_a.\mathsf{obs} \cup G_a.\mathsf{dob} \cup G_a.\mathsf{aob} \cup G_a.\mathsf{bob}$ is acyclic. (EXTERNAL)

We interpret the intended compilation on execution graphs:

Definition 3.2. Let G be an IMM execution graph with whole serial numbers ($\mathsf{sn}[G.\mathsf{E}] \subseteq \mathbb{N}$). An ARM execution graph G_a *corresponds* to G if the following hold:

- $G_a.\mathsf{E} = G.\mathsf{E} \cup \{\langle i, n+0.5\rangle \mid \langle i, n\rangle \in G.\mathsf{W}_{\mathtt{strong}}\}$
  (new events are added after strong exclusive writes)
- $G_a.\mathtt{lab} = \{e \mapsto (\!|G.\mathtt{lab}(e)|\!) \mid e \in G.\mathsf{E}\} \cup \{e \mapsto \mathsf{F}^{\mathsf{ld}} \mid e \in G_a.\mathsf{E} \setminus G.\mathsf{E}\}$ where:

$$(\!|\mathsf{R}_s^{\mathtt{rlx}}(x,v)|\!) \triangleq \mathsf{R}^{\mathtt{rlx}}(x,v) \qquad (\!|\mathsf{W}_{o_{\mathsf{RMW}}}^{\mathtt{rlx}}(x,v)|\!) \triangleq \mathsf{W}^{\mathtt{rlx}}(x,v)$$
$$(\!|\mathsf{R}_s^{\mathtt{acq}}(x,v)|\!) \triangleq \mathsf{R}^{\mathsf{Q}}(x,v) \qquad (\!|\mathsf{W}_{o_{\mathsf{RMW}}}^{\sqsupseteq \mathtt{rel}}(x,v)|\!) \triangleq \mathsf{W}^{\mathsf{L}}(x,v)$$
$$(\!|\mathsf{R}_s^{\mathtt{sc}}(x,v)|\!) \triangleq \mathsf{R}^{\mathsf{A}}(x,v)$$
$$(\!|\mathsf{F}^{\mathtt{acq}}|\!) \triangleq \mathsf{F}^{\mathsf{ld}} \qquad (\!|\mathsf{F}^{\mathtt{rel}}|\!) = (\!|\mathsf{F}^{\mathtt{acqrel}}|\!) = (\!|\mathsf{F}^{\mathtt{sc}}|\!) \triangleq \mathsf{F}^{\mathsf{sy}}$$

- $G.\mathtt{rmw} = G_a.\mathtt{rmw}$, $G.\mathtt{data} = G_a.\mathtt{data}$, and $G.\mathtt{addr} = G_a.\mathtt{addr}$
  (the compilation does not change RMW pairs and data/address dependencies)
- $G.\mathtt{ctrl} \subseteq G_a.\mathtt{ctrl}$
  (the compilation only adds control dependencies)
- $[G.\mathsf{R}_{\mathtt{ex}}]\,;G.\mathsf{po} \subseteq G_a.\mathtt{ctrl} \cup G_a.\mathtt{rmw} \cap G_a.\mathtt{data}$
  (exclusive reads entail a control dependency to any future event, except for their immediate exclusive write successor if arose from an atomic increment)
- $G.\mathtt{casdep}\,;G.\mathsf{po} \subseteq G_a.\mathtt{ctrl}$
  (CAS dependency to an exclusive read entails a control dependency to any future event)

We state our theorem that ensures $\mathsf{IMM}_{\mathsf{SC}}$-consistency if the corresponding ARMv8 execution graph is ARMv8-consistent.

THEOREM 3.3. *Let G be an* IMM *execution graph with whole serial numbers ($\mathsf{sn}[G.\mathsf{E}] \subseteq \mathbb{N}$), and let G_a be an* ARMv8 *execution graph that corresponds to G. Then,* ARMv8*-consistency of G_a implies* $\mathsf{IMM}_{\mathsf{SC}}$*-consistency of G.*

PROOF (OUTLINE). IMM-consistency of G follows from [Podkopaev et al. 2019, Theorem 4.5]. That is, we only need to show that the sc axiom holds for G. We start by showing that $G_a.\mathsf{obs}' \cup G_a.\mathsf{dob} \cup G_a.\mathsf{aob} \cup G_a.\mathsf{bob}'$ is acyclic, where

$$\mathsf{obs}' \triangleq \mathtt{rfe} \cup \mathtt{fr} \cup \mathtt{co}$$
$$\mathsf{bob}' \triangleq \mathsf{bob} \cup [\mathsf{R}];\mathsf{po};[\mathsf{F}^{\mathsf{ld}}] \cup \mathsf{po};[\mathsf{F}^{\mathsf{sy}}] \cup [\mathsf{F}^{\sqsupseteq \mathsf{ld}}];\mathsf{po}$$

Then, we finish the proof by showing that $G_a.\mathsf{psc}_{\mathsf{base}} \cup G_a.\mathsf{psc}_{\mathsf{F}}$ is included in $(G_a.\mathsf{obs}' \cup G_a.\mathsf{dob} \cup G_a.\mathsf{aob} \cup G_a.\mathsf{bob}')^+$. □

$(\!\|r := [e]^{\neq \mathsf{SC}}\|\!) \approx$ "mov"	$(\!\|\mathbf{fence}^{\neq \mathsf{SC}}\|\!) \approx$ ""
$(\!\|[e_1]^{\neq \mathsf{SC}} := e_2\|\!) \approx$ "mov"	$(\!\|\mathbf{fence}^{\mathsf{SC}}\|\!) \approx$ "mfence"
$(\!\|r := \mathbf{FADD}^{o_R, o_W}_{o_{RMW}}(e_1, e_2)\|\!) \approx$ "(lock) xchg"	$(\!\|r := \mathbf{CAS}^{o_R, o_W}_{o_{RMW}}(e, e_R, e_W)\|\!) \approx$ "(lock) xchg"
Alt. 1: $(\!\|r := [e]^{\mathsf{SC}}\|\!) \approx$ "mov"	**Alt. 2:** $(\!\|r := [e]^{\mathsf{SC}}\|\!) \approx$ "mfence;mov"
$(\!\|[e_1]^{\mathsf{SC}} := e_2\|\!) \approx$ "mov;mfence"	$(\!\|[e_1]^{\mathsf{SC}} := e_2\|\!) \approx$ "mov"

Fig. 2. Compilation scheme from $\mathsf{IMM}_{\mathsf{SC}}$ to TSO.

4 FROM $\mathsf{IMM}_{\mathsf{SC}}$ TO TSO

The intended mapping of $\mathsf{IMM}_{\mathsf{SC}}$ to TSO is presented schematically in Fig. 2. There are two possible alternatives for compiling SC accesses (see the bottom of Fig. 2): to compile an SC store to a store followed by a fence or to compile an SC load to a load preceded by a fence. Both of the schemes guarantee that in compiled code there is a fence between every store and load instructions originated from SC accesses. Regarding compilation schemes of SC accesses, our proof of the compilation correctness from $\mathsf{IMM}_{\mathsf{SC}}$ to TSO depends only on this property. That is, in this section, we concentrate only on the compilation alternative which compiles SC stores using fences.

As a model of the TSO architecture, we use a declarative model from [Alglave et al. 2014]. Its labels are given by:

- TSO read label: $\mathtt{R}(x, v)$ where $x \in \mathsf{Loc}$ and $v \in \mathsf{Val}$.
- TSO write label: $\mathtt{W}(x, v)$ where $x \in \mathsf{Loc}$ and $v \in \mathsf{Val}$.
- TSO fence label: MFENCE.

In turn, TSO's execution graphs are defined as $\mathsf{IMM}_{\mathsf{SC}}$'s ones. Below, we interpret the compilation on execution graphs.

Definition 4.1. Let G be an IMM execution graph with whole serial numbers ($\mathsf{sn}[G.\mathsf{E}] \subseteq \mathbb{N}$). A TSO execution graph G_t *corresponds* to G if the following hold:

- $G_t.\mathsf{E} = G.\mathsf{E} \setminus G.\mathsf{F}^{\neq \mathsf{SC}} \cup \{\langle i, n + 0.5\rangle \mid \langle i, n\rangle \in G.\mathsf{W}^{\mathsf{SC}}\}$
  (non-SC fences are removed and new events are added after SC writes)
- $G_t.\mathtt{lab} = \{e \mapsto (\!|G.\mathtt{lab}(e)|\!) \mid e \in G.\mathsf{E} \setminus G.\mathsf{F}^{\neq \mathsf{SC}}\} \cup \{e \mapsto \mathtt{MFENCE} \mid e \in G_t.\mathsf{E} \setminus G.\mathsf{E}\}$ where:

$$(\!|\mathsf{R}^{o_R}_s(x, v)|\!) \triangleq \mathtt{R}(x, v) \qquad (\!|\mathsf{W}^{o_W}_{o_{RMW}}(x, v)|\!) \triangleq \mathtt{W}(x, v) \qquad (\!|\mathsf{F}^{\mathsf{SC}}|\!) \triangleq \mathtt{MFENCE}$$

- $G.\mathtt{rmw} = G_t.\mathtt{rmw}$, $G.\mathtt{data} = G_t.\mathtt{data}$, and $G.\mathtt{addr} = G_t.\mathtt{addr}$
  (the compilation does not change RMW pairs and data/address dependencies)
- $G.\mathtt{ctrl}; [G.\mathsf{E} \setminus G.\mathsf{F}^{\neq \mathsf{SC}}] \subseteq G_t.\mathtt{ctrl}$
  (the compilation only adds control dependencies)

The following derived relations are used to define the TSO-consistency predicate.

$$\begin{aligned}
\mathsf{ppo}_{\mathsf{TSO}} &\triangleq [\mathtt{RW}]; \mathsf{po}; [\mathtt{RW}] \setminus [\mathtt{W}]; \mathsf{po}; [\mathtt{R}] \\
\mathtt{fence}_{\mathsf{TSO}} &\triangleq [\mathtt{RW}]; \mathsf{po}; [\mathtt{MFENCE}]; \mathsf{po}; [\mathtt{RW}] \\
\mathtt{implied_fence}_{\mathsf{TSO}} &\triangleq [\mathtt{W}]; \mathsf{po}; [\mathit{dom}(\mathtt{rmw})] \cup [\mathit{codom}(\mathtt{rmw})]; \mathsf{po}; [\mathtt{R}] \\
\mathsf{hb}_{\mathsf{TSO}} &\triangleq \mathsf{ppo}_{\mathsf{TSO}} \cup \mathtt{fence}_{\mathsf{TSO}} \cup \mathtt{implied_fence}_{\mathsf{TSO}} \cup \mathtt{rfe} \cup \mathtt{co} \cup \mathtt{fr}
\end{aligned}$$

Definition 4.2. G is called TSO-*consistent* if the following hold:

- $\mathit{codom}(G.\mathtt{rf}) = G.\mathsf{R}$. (rf-completeness)
- For every location $x \in \mathsf{Loc}$, $G.\mathtt{co}$ totally orders $G.\mathtt{W}(x)$. (co-totality)
- $\mathsf{po}|_{\mathtt{loc}} \cup \mathtt{rf} \cup \mathtt{fr} \cup \mathtt{co}$ is acyclic. (sc-per-loc)
- $G.\mathtt{rmw} \cap (G.\mathtt{fre}\ ;\ G.\mathtt{coe}) = \emptyset$. (atomicity)

$$(\!|r := [e]^{\mathsf{rlx}}|\!) \approx \text{“ld”} \qquad (\!|[e_1]^{\mathsf{rlx}} := e_2|\!) \approx \text{“st”}$$
$$(\!|r := [e]^{\mathsf{acq}}|\!) \approx \text{“ld;cmp;bc;isync”} \qquad (\!|[e_1]^{\mathsf{rel}} := e_2|\!) \approx \text{“lwsync;st”}$$
$$(\!|\mathbf{fence}^{\neq \mathsf{sc}}|\!) \approx \text{“lwsync”} \qquad (\!|\mathbf{fence}^{\mathsf{sc}}|\!) \approx \text{“sync”}$$
$$(\!|r := \mathbf{FADD}^{o_{\mathsf{R}},o_{\mathsf{W}}}_{o_{\mathsf{RMW}}}(e_1,e_2)|\!) \approx \mathrm{wmod}(o_{\mathsf{W}}) +\!\!+ \text{“L:lwarx;stwcx.;bc L”} +\!\!+ \mathrm{rmod}(o_{\mathsf{R}})$$
$$(\!|r := \mathbf{CAS}^{o_{\mathsf{R}},o_{\mathsf{W}}}_{o_{\mathsf{RMW}}}(e,e_{\mathsf{R}},e_{\mathsf{W}})|\!) \approx \mathrm{wmod}(o_{\mathsf{W}}) +\!\!+ \text{“L:lwarx;cmp;bc Le;stwcx.;bc L;Le:”} +\!\!+ \mathrm{rmod}(o_{\mathsf{R}})$$
$$\mathrm{wmod}(o_{\mathsf{W}}) \triangleq o_{\mathsf{W}} = \mathtt{rel}\ ?\ \text{“lwsync;”}\ :\ \text{“”} \qquad \mathrm{rmod}(o_{\mathsf{R}}) \triangleq o_{\mathsf{R}} = \mathtt{acq}\ ?\ \text{“;isync”}\ :\ \text{“”}$$

Leading sync:	**Trailing** sync:
$(\!\|r := [e]^{\mathsf{sc}}\|\!) \approx$ “sync;ld;cmp;bc;isync”	$(\!\|r := [e]^{\mathsf{sc}}\|\!) \approx$ “ld;sync”
$(\!\|[e_1]^{\mathsf{sc}} := e_2\|\!) \approx$ “sync;st”	$(\!\|[e_1]^{\mathsf{sc}} := e_2\|\!) \approx$ “lwsync;st;sync”

Fig. 3. Compilation scheme from IMM to POWER.

- $G.\mathsf{hb}_{\mathsf{TSO}}$ is acyclic. (TSO-NO-THIN-AIR)

Next, we state our theorem that ensures $\mathsf{IMM}_{\mathsf{SC}}$-consistency if the corresponding TSO execution graph is TSO-consistent.

Theorem 4.3. *Let G be an $\mathsf{IMM}_{\mathsf{SC}}$ execution graph with whole serial numbers ($\mathsf{sn}[G.\mathsf{E}] \subseteq \mathbb{N}$), and let G_t be an TSO execution graph that corresponds to G. Then, TSO-consistency of G_t implies $\mathsf{IMM}_{\mathsf{SC}}$-consistency of G.*

Proof (Outline). Since G_t corresponds to G, we know that

$$[G.\mathsf{W}^{\mathsf{sc}}]; G.\mathsf{po}; [G.\mathsf{R}^{\mathsf{sc}}] \subseteq G_t.\mathsf{po}; [G_t.\mathtt{MFENCE}]; G_t.\mathsf{po}$$

as the aforementioned property of the compilation scheme. We show that

$$G_t.\mathsf{ehb}_{\mathsf{TSO}} \triangleq G_t.\mathsf{hb}_{\mathsf{TSO}} \cup [G_t.\mathtt{MFENCE}]; G_t.\mathsf{po} \cup [G_t.\mathtt{MFENCE}]; G_t.\mathsf{po}$$

is acyclic. Then, we show that $G.\mathsf{psc}_{\mathsf{base}} \cup G.\mathsf{psc}_{\mathsf{F}}$ is included in $G_t.\mathsf{ehb}^{+}_{\mathsf{TSO}}$. It means that the sc axiom holds for G, and it leaves us to prove that G is IMM-consistent. That is done by standard relational techniques (see [Coq proofs 2019]). □

5 FROM $\mathsf{IMM}_{\mathsf{SC}}$ TO POWER

Here we use the same mapping of IMM to POWER (see Fig. 3) as in [Podkopaev et al. 2019] for all instructions except for SC accesses. For the latter, there are two standard compilations schemes [Mapping 2016] presented in the bottom of Fig. 3: with leading and trailing sync fences.

Podkopaev et al. [2019] prove IMM to POWER compilation correctness in two steps. First, on the side of IMM, they prove correctness of splitting a release write to a relaxed write and a preceding release fence. Such splitting corresponds to the compilation scheme for release writes. Second, they prove correctness of compilation from IMM to POWER for programs without release writes.

In our proof we introduce a preceding step. At this step we prove that splitting of every SC read and write to a pair of an SC fence (which corresponds to sync instruction in POWER) and either an acquire read or a release write respectively is sound in $\mathsf{IMM}_{\mathsf{SC}}$. The splitting should follow either leading or trailing compilation scheme. Assuming that SC accesses are removed from a program by the splitting, we can reuse the compilation result from [Podkopaev et al. 2019].

Theorem 5.1. *Let G be an IMM execution graph such that*

$$[G.\mathsf{RW}^{\mathsf{sc}}]; (G.\mathsf{po}' \cup G.\mathsf{po}'; G.\mathsf{hb}; G.\mathsf{po}'); [G.\mathsf{RW}^{\mathsf{sc}}] \subseteq G.\mathsf{hb}; [G.\mathsf{F}^{\mathsf{sc}}]; G.\mathsf{hb},$$

where $G.\mathsf{po}' \triangleq G.\mathsf{po} \setminus G.\mathtt{rmw}$. Let G' be the IMM execution graph obtained from G by weakening the access modes of SC write and read events to release and acquire modes respectively. Then, $\mathsf{IMM}_{\mathsf{SC}}$-consistency of G follows from IMM-consistency of G'.

Note that in Theorem 5.1 we assume neither leading nor trailing compilation schemes explicitly but require to have an SC fence on some hb paths between SC accesses. Both schemes provide this.

In a compiler, direct following of the splitting steps for SC accesses and release writes leads to redundant instructions in generated code. Thus, imagine such a compiler which uses the leading compilation scheme for SC accesses. It would emit `sync;lwsync;st` for an SC write. Clearly, `lwsync` is redundant here since `sync` provides stronger guarantees than `lwsync`. To avoid it, we assume that a release write splitting produces a release fence only if necessary, *i.e.,* the release write is not produced on a preceding step by splitting an SC write. It does not break the release splitting result [Podkopaev et al. 2019, Theorem 4.1] since it only requires presence of a release or SC fence on any po path to a release write for the execution graph after the split.[2]

In case a compiler uses the trailing compilation scheme, an analogous redundancy arises for SC reads: they are compiled to `ld;cmp;bc;isync;sync`. Here `sync` makes other synchronization instructions redundant. To use the same trick as with SC writes in case of the leading compilation scheme, we extended the compilation result from [Podkopaev et al. 2019] by showing that `ld;lwsync` (as well as a stronger version `ld;sync`) is a correct compilation scheme for acquire reads.

The next definition presents the correspondence between IMM execution graphs and their mapped POWER ones following the leading compilation scheme in Fig. 3 with elimination of the aforementioned redundancy of SC write compilation.

Definition 5.2. Let G be an IMM execution graph with whole positive serial numbers ($\mathsf{sn}[G.\mathsf{E}] \subseteq \mathbb{N}^+$). A POWER execution graph G_p *corresponds* to G if the following hold:

- $G_p.\mathsf{E} = G.\mathsf{E} \cup \{\langle i, n+0.3\rangle \mid \langle i,n\rangle \in (G.\mathsf{R}^{\sqsupseteq \mathsf{acq}} \setminus dom(G.\mathtt{rmw})) \cup codom([G.\mathsf{R}^{\sqsupseteq \mathsf{acq}}]\,;G.\mathtt{rmw})\}$
  $\cup\ \{\langle i, n-0.3\rangle \mid \langle i,n\rangle \in (G.\mathsf{E}^{\sqsupseteq \mathsf{rel}} \setminus dom(G.\mathtt{rmw})) \cup dom(G.\mathtt{rmw}\,;[G.\mathsf{W}^{\sqsupseteq \mathsf{rel}}])\}$
  (new events are added after acquire reads and acquire RMW pairs and before SC accesses and SC RMW pairs)
- $G_p.\mathtt{lab} = \{e \mapsto (\!|G.\mathtt{lab}(e)|\!) \mid e \in G.\mathsf{E}\} \cup$
  $\{\langle i, n+0.3\rangle \mapsto \mathsf{F}^{\mathsf{isync}} \mid \langle i, n+0.3\rangle \in G_p.\mathsf{E} \wedge n \in \mathbb{N}^+\} \cup$
  $\{\langle i, n-0.3\rangle \mapsto \mathsf{F}^{\mathsf{lwsync}} \mid \langle i, n-0.3\rangle \in G_p.\mathsf{E} \wedge n \in \mathbb{N}^+ \wedge \langle i,n\rangle \notin G.\mathsf{E}^{\mathsf{sc}} \cup dom(G.\mathtt{rmw}\,;[G.\mathsf{W}^{\mathsf{sc}}])\} \cup$
  $\{\langle i, n-0.3\rangle \mapsto \mathsf{F}^{\mathsf{sync}} \mid \langle i, n-0.3\rangle \in G_p.\mathsf{E} \wedge n \in \mathbb{N}^+ \wedge \langle i,n\rangle \in G.\mathsf{E}^{\mathsf{sc}} \cup dom(G.\mathtt{rmw}\,;[G.\mathsf{W}^{\mathsf{sc}}])\}$
  where:
  $$(\!|\mathsf{R}^{o_\mathsf{R}}_s(x,v)|\!) \triangleq \mathsf{R}(x,v) \qquad (\!|\mathsf{F}^{\mathsf{acq}}|\!) = (\!|\mathsf{F}^{\mathsf{rel}}|\!) = (\!|\mathsf{F}^{\mathsf{acqrel}}|\!) \triangleq \mathsf{F}^{\mathsf{lwsync}}$$
  $$(\!|\mathsf{W}^{o_\mathsf{W}}_{o_{\mathsf{RMW}}}(x,v)|\!) \triangleq \mathsf{W}(x,v) \qquad (\!|\mathsf{F}^{\mathsf{sc}}|\!) \triangleq \mathsf{F}^{\mathsf{sync}}$$
- $G.\mathtt{rmw} = G_p.\mathtt{rmw}$, $G.\mathtt{data} = G_p.\mathtt{data}$, and $G.\mathtt{addr} = G_p.\mathtt{addr}$
  (the compilation does not change RMW pairs and data/address dependencies)
- $G.\mathtt{ctrl} \subseteq G_p.\mathtt{ctrl}$
  (the compilation only adds control dependencies)
- $[G.\mathsf{R}^{\sqsupseteq \mathsf{acq}}]\,;G.\mathsf{po} \subseteq G_p.\mathtt{rmw} \cup G_p.\mathtt{ctrl}$
  (a control dependency is placed from every acquire or SC read)
- $[G.\mathsf{R}_{\mathsf{ex}}]\,;G.\mathsf{po} \subseteq G_p.\mathtt{ctrl} \cup G_p.\mathtt{rmw} \cap G_p.\mathtt{data}$
  (exclusive reads entail a control dependency to any future event, except for their immediate exclusive write successor if arose from an atomic increment)

[2] Actually, [Podkopaev et al. 2019, Theorem 4.1] requires a target execution graph to have exactly a release fence (not SC one) after a release write. However, such strict requirement is not necessary, and the corresponding Coq proofs [2019] allow SC fences in this context.

- $G.\mathtt{data} ; [\mathit{codom}(G.\mathtt{rmw})] ; G.\mathsf{po} \subseteq G_p.\mathtt{ctrl}$
  (data dependency to an exclusive write entails a control dependency to any future event)
- $G.\mathtt{casdep} ; G.\mathsf{po} \subseteq G_p.\mathtt{ctrl}$
  (CAS dependency to an exclusive read entails a control dependency to any future event)

The correspondence between IMM and POWER execution graphs which follows the trailing compilation scheme may be presented similarly with two main difference. First, obviously, SC accesses are compiled to release and acquire accesses followed by SC fences:

$$\begin{aligned} G_p.\mathsf{E} = G.\mathsf{E} \cup \{\langle i, n+0.3\rangle \mid \langle i,n\rangle \in (G.\mathsf{E}^{\sqsupseteq \mathtt{acq}} \setminus \mathit{dom}(G.\mathtt{rmw})) \cup \mathit{codom}([G.\mathsf{R}^{\sqsupseteq \mathtt{acq}}] ; G.\mathtt{rmw})\} \\ \cup \{\langle i, n-0.3\rangle \mid \langle i,n\rangle \in (G.\mathsf{W}^{\sqsupseteq \mathtt{rel}} \setminus \mathit{dom}(G.\mathtt{rmw})) \cup \mathit{dom}(G.\mathtt{rmw} ; [G.\mathsf{W}^{\sqsupseteq \mathtt{rel}}])\} \end{aligned}$$

$$\begin{aligned} G_p.\mathtt{lab} = \ & \{e \mapsto (\!|G.\mathtt{lab}(e)|\!) \mid e \in G.\mathsf{E}\} \cup \\ & \{\langle i, n+0.3\rangle \mapsto \mathsf{F}^{\mathtt{isync}} \mid \langle i, n+0.3\rangle \in G_p.\mathsf{E} \wedge n \in \mathbb{N}^+ \wedge \\ & \qquad\qquad \langle i,n\rangle \in G.\mathsf{R}^{\mathtt{acq}} \cup \mathit{codom}([G.\mathsf{R}^{\mathtt{acq}}] ; G.\mathtt{rmw})\} \cup \\ & \{\langle i, n+0.3\rangle \mapsto \mathsf{F}^{\mathtt{sync}} \mid \langle i, n+0.3\rangle \in G_p.\mathsf{E} \wedge n \in \mathbb{N}^+ \wedge \\ & \qquad\qquad \langle i,n\rangle \in G.\mathsf{E}^{\mathtt{sc}} \cup \mathit{codom}([G.\mathsf{R}^{\mathtt{sc}}] ; G.\mathtt{rmw})\} \cup \\ & \{\langle i, n-0.3\rangle \mapsto \mathsf{F}^{\mathtt{lwsync}} \mid \langle i, n-0.3\rangle \in G_p.\mathsf{E} \wedge n \in \mathbb{N}^+\} \end{aligned}$$

Second, $[G.\mathsf{R}^{\sqsupseteq \mathtt{acq}}] ; G.\mathsf{po}$ has to be included in $G_p.\mathtt{rmw} \cup G_p.\mathtt{ctrl} \cup G_p.\mathsf{po} ; [G_p.\mathsf{F}^{\mathtt{lwsync}}] ; G_p.\mathsf{po}^?$, not just in $G_p.\mathtt{rmw} \cup G_p.\mathtt{ctrl}$, to allow for elimination of the aforementioned SC read compilation redundancy.

The next theorem ensures $\mathsf{IMM}_{\mathsf{SC}}$-consistency if the corresponding POWER execution graph is POWER-consistent.

Theorem 5.3. *Let G be an* IMM *execution graph with whole serial numbers ($\mathsf{sn}[G.\mathsf{E}] \subseteq \mathbb{N}$), and let G_p be a* POWER *execution graph that corresponds to G. Then,* POWER*-consistency of G_p implies* $\mathsf{IMM}_{\mathsf{SC}}$*-consistency of G.*

Proof (Outline). We construct an IMM execution graph G' by inserting SC fences before SC accesses in G. We also construct $G_{\mathtt{NoSC}}$ from G' by replacing SC write and read accesses of G' with release write and acquire read ones respectively. Obviously, $\mathsf{IMM}_{\mathsf{SC}}$-consistency of G follows from $\mathsf{IMM}_{\mathsf{SC}}$-consistency of G', which, in turn, follows from IMM-consistency of $G_{\mathtt{NoSC}}$ by Theorem 5.1. We construct an IMM execution graph G'' from $G_{\mathtt{NoSC}}$ by inserting release fences before release writes, and then an IMM execution graph $G_{\mathtt{NoRel}}$ from G'' by weakening the access modes of release write events to a relaxed mode. As on a previous proof step, IMM-consistency of $G_{\mathtt{NoSC}}$ follows from IMM-consistency of G'', which in turn follows from IMM-consistency of $G_{\mathtt{NoRel}}$ by [Podkopaev et al. 2019, Theorem 4.1].

Thus to prove the theorem we need to show that $G_{\mathtt{NoRel}}$ is IMM-consistent. Note that G_p—the POWER execution graph corresponding to G—also corresponds to $G_{\mathtt{NoRel}}$ by construction of $G_{\mathtt{NoRel}}$. That is, IMM-consistency of $G_{\mathtt{NoRel}}$ follows from POWER-consistency of G_p by [Podkopaev et al. 2019, Theorem 4.3] since $G_{\mathtt{NoRel}}$ does not contain SC read and write access events as well as release write access events. □

ACKNOWLEDGMENTS

The first author was supported by JetBrains Research and RFBR (grant number 18-01-00380). The second author was supported by the Israel Science Foundation (grant number 5166651), and by Len Blavatnik and the Blavatnik Family foundation.

REFERENCES

Jade Alglave, Luc Maranget, and Michael Tautschnig. 2014. Herding Cats: Modelling, Simulation, Testing, and Data Mining for Weak Memory. *ACM Trans. Program. Lang. Syst.* 36, 2, Article 7 (July 2014), 74 pages. https://doi.org/10.1145/2627752

Mark Batty, Scott Owens, Susmit Sarkar, Peter Sewell, and Tjark Weber. 2011. Mathematizing C++ Concurrency. In *POPL 2011*. ACM, New York, 55–66. https://doi.org/10.1145/1925844.1926394

Coq proofs 2019. Coq proof scripts for this paper, available at http://github.com/weakmemory/imm.

Will Deacon. 2017. The ARMv8 Application Level Memory Model. Retrieved June 27, 2018 from https://github.com/herd/herdtools7/blob/master/herd/libdir/aarch64.cat

Ori Lahav, Viktor Vafeiadis, Jeehoon Kang, Chung-Kil Hur, and Derek Dreyer. 2017. Repairing Sequential Consistency in C/C++11. In *PLDI 2017*. ACM, New York, 618–632. https://doi.org/10.1145/3062341.3062352

Mapping 2016. C/C++11 mappings to processors. Retrieved June 27, 2018 from http://www.cl.cam.ac.uk/~pes20/cpp/cpp0xmappings.html

Anton Podkopaev, Ori Lahav, and Viktor Vafeiadis. 2019. Bridging the gap between programming languages and hardware weak memory models. *Proc. ACM Program. Lang.* 3, POPL (2019), 69:1–69:31. https://doi.org/10.1145/3290382

Christopher Pulte, Shaked Flur, Will Deacon, Jon French, Susmit Sarkar, and Peter Sewell. 2018. Simplifying ARM concurrency: multicopy-atomic axiomatic and operational models for ARMv8. *Proc. ACM Program. Lang.* 2, POPL (2018), 19:1–19:29. https://doi.org/10.1145/3158107

Viktor Vafeiadis, Thibaut Balabonski, Soham Chakraborty, Robin Morisset, and Francesco Zappa Nardelli. 2015. Common Compiler Optimisations are Invalid in the C11 Memory Model and what we can do about it. In *POPL 2015*. ACM, New York, 209–220. https://doi.org/10.1145/2676726.2676995