

Program verification under weak memory consistency

Viktor Vafeiadis

MPI-SWS

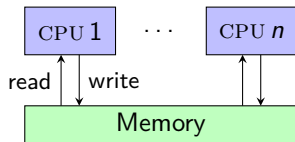
Marktobersdorf, August 2019

Weak memory consistency
is about the
semantics of concurrent programs
taking into account the effects of:

- ▶ multicore **hardware** implementations
- ▶ and **compiler** optimizations.

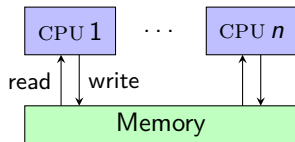
Sequential consistency (SC)

- ▶ The standard simplistic concurrency model.
- ▶ Threads access shared memory in an interleaved fashion.



Sequential consistency (SC)

- ▶ The standard simplistic concurrency model.
- ▶ Threads access shared memory in an interleaved fashion.



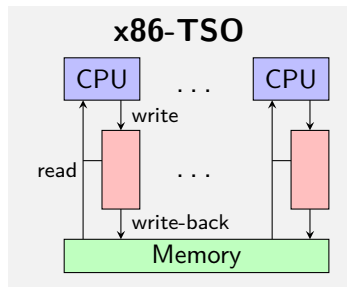
But...

- ▶ No multicore processor implements SC.
- ▶ Compiler optimizations invalidate SC.
- ▶ In most cases, SC is not really necessary.

Store buffering (SB)

Initially, $x = y = 0$

$x := 1;$ $y := 1;$
 $a := y$ //0 $b := x$ //0



Store buffering (SB)

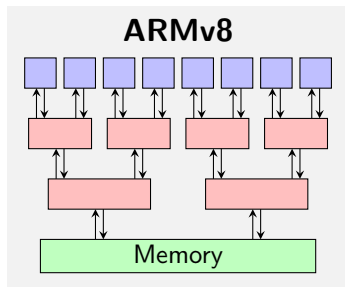
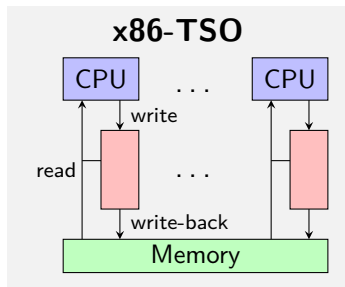
Initially, $x = y = 0$

$x := 1;$ \parallel $y := 1;$
 $a := y$ *//0* \parallel $b := x$ *//0*

Load buffering (LB)

Initially, $x = y = 0$

$a := y;$ *//1* \parallel $b := x;$ *//1*
 $x := 1$ \parallel $y := 1$



- ▶ Messages may be delayed.



$MsgX := 1;$
 $a := MsgY; //0$



$MsgY := 1;$
 $b := MsgX; //0$



Weak consistency in “real life”

- ▶ Messages may be delayed.


$$\begin{array}{l} \textit{MsgX} := 1; \\ a := \textit{MsgY}; \textit{//0} \end{array} \parallel \begin{array}{l} \textit{MsgY} := 1; \\ b := \textit{MsgX}; \textit{//0} \end{array}$$


- ▶ Messages may be sent/received out of order.

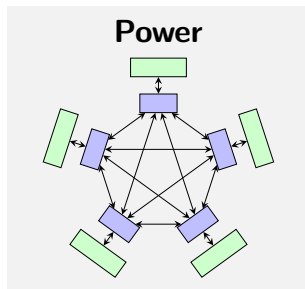

$$\begin{array}{l} \textit{Email} := 1; \\ \textit{Sms} := 1; \end{array} \parallel \begin{array}{l} a := \textit{Sms}; \textit{//1} \\ b := \textit{Email}; \textit{//0} \end{array}$$


Independent reads of independent writes (IRIW)

Initially, $x = y = 0$

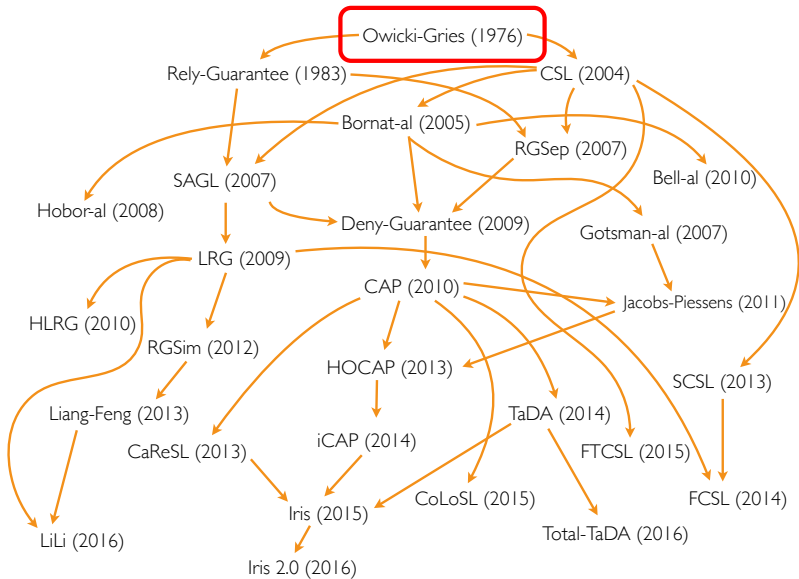
$$x := 1 \quad \left\| \begin{array}{l} a := x; \text{ //1} \\ \text{lwsync;} \\ b := y \text{ //0} \end{array} \right\| \left\| \begin{array}{l} c := y; \text{ //1} \\ \text{lwsync;} \\ d := x \text{ //0} \end{array} \right\| \quad y := 1$$

- ▶ Thread II and III can observe the $x := 1$ and $y := 1$ writes happen in different orders.
- ▶ Because of the `lwsync` fences, no reorderings are possible!



Standard proof techniques
are **unsound** under
weak memory consistency

e.g., Owicki-Gries



Program logics for concurrent programs (adapted from Ilya Sergey)

$$\{P\} c \{Q\}$$

► P : precondition

► c : program

► Q : postcondition

$$\overline{\{P\} \text{ skip } \{P\}}$$

$$\overline{\{P[e/x]\} x := e \{P\}}$$

$$\frac{\{P\} c_1 \{R\} \quad \{R\} c_2 \{Q\}}{\{P\} c_1; c_2 \{Q\}}$$

$$\frac{\begin{array}{l} \{e \neq 0 \wedge P\} c_1 \{Q\} \\ \{e = 0 \wedge P\} c_2 \{Q\} \end{array}}{\{P\} \text{ if } e \text{ then } c_1 \text{ else } c_2 \{Q\}}$$

$$\frac{\{P \wedge e \neq 0\} c \{P\}}{\{P\} \text{ while } e \text{ do } c \{P \wedge e = 0\}}$$

$$\frac{P_1 \Rightarrow P_2 \quad \{P_2\} c \{Q_2\} \quad Q_2 \Rightarrow Q_1}{\{P_1\} c \{Q_1\}}$$

OG = Hoare logic + rule for parallel composition

$$\frac{\{P_1\} c_1 \{Q_1\} \quad \{P_2\} c_2 \{Q_2\} \quad \text{the two proofs are } \textit{non-interfering}}{\{P_1 \wedge P_2\} c_1 \parallel c_2 \{Q_1 \wedge Q_2\}}$$

Non-interference

$R \wedge P \vdash R[u/x]$ for every:

- ▶ assertion R in one proof outline
- ▶ assignment $x := u$ with precondition P in the other proof outline

$$\begin{array}{c} \vdots \\ \{P\} \\ x := u \\ \vdots \end{array} \parallel \begin{array}{c} \vdots \\ \{R\} \\ \vdots \end{array}$$

Example SB: store buffering

$$\begin{array}{c} \{a \neq 0\} \\ \{a \neq 0\} \\ x := 1 \\ \{x \neq 0\} \\ a := y \\ \{x \neq 0\} \\ \{a \neq 0 \vee b \neq 0\} \end{array} \parallel \begin{array}{c} \{a \neq 0\} \\ \{\top\} \\ y := 1 \\ \{y \neq 0\} \\ b := x \\ \{y \neq 0 \wedge (a \neq 0 \vee b = x)\} \end{array}$$

Example SB: store buffering

$$\begin{array}{c} \{a \neq 0\} \\ \{a \neq 0\} \\ x := 1 \\ \{x \neq 0\} \\ a := y \\ \{x \neq 0\} \\ \{a \neq 0 \vee b \neq 0\} \end{array} \parallel \begin{array}{c} \{a \neq 0\} \\ \{\top\} \\ y := 1 \\ \{y \neq 0\} \\ b := x \\ \{y \neq 0 \wedge (a \neq 0 \vee b = x)\} \end{array}$$

Example SB: store buffering

$$\begin{array}{c} \{a \neq 0\} \\ x := 1 \\ \{x \neq 0\} \\ a := y \\ \{x \neq 0\} \\ \{a \neq 0 \vee b \neq 0\} \end{array} \parallel \begin{array}{c} \{a \neq 0\} \\ \{\top\} \\ y := 1 \\ \{y \neq 0\} \\ b := x \\ \{y \neq 0 \wedge (a \neq 0 \vee b = x)\} \end{array}$$

Standard OG is **unsound** under weak memory!

Other problems
are **easier** under
weak memory consistency

e.g., execution consistency

Execution consistency problem

Given a concurrent program P with instructions of the form:

- ▶ $x := v$ – write constant v to shared variable x
- ▶ $r := x$ – read value of x into register r

such that no two instructions have the same v or r , and a register assignment R , determine whether R is a possible outcome of P .

This problem is:

- ▶ NP-complete for SC;
- ▶ Polynomial for several weak memory models (e.g., RA).

NB: There are other verification problems that are easy under SC and difficult under WMC.

Weak consistency is not a threat, but an opportunity.

- ▶ Can lead to more scalable concurrent algorithms.
- ▶ Several open research problems.

Reasoning under WMC is often easier than under SC.

- ▶ Avoid thinking about thread interleavings.
- ▶ Many concurrent algorithms do not need SC!