Introduction to weak memory consistency

Ori Lahav    Viktor Vafeiadis

28 August 2017
Weak memory consistency is about the semantics of concurrent programs taking into account the effects of:

- multicore hardware implementations
- and compiler optimizations.
CPU trends: Parallelism is here!

![Graph showing CPU trends with single-core and multi-core eras]

- Single-Thread Performance (SpecINT x 10²)
- Number of Logical Cores
Concurrent programming is hard!

If you can get away with it, avoid using threads. Threads can be difficult to use, and they make programs harder to debug.

(Java documentation, ≈ 15 years ago)

“Difficult to use”

- Requires a fundamentally different way of thinking.
- Interference among threads.

“Harder to debug”

- Huge non-determinism ↼ testing is ineffective.

\[
X := X + 1 \quad \| \quad X := X + 1 \text{ might increment } X \text{ only once.}
\]

Thread 1: \[\text{Read } X=0 \quad \text{Write } X=1\]

Thread 2: \[\text{Read } X=0 \quad \text{Write } X=1\]
The illusion of sequential consistency

Sequential consistency (SC)
- The standard simplistic concurrency model.
- Threads access shared memory in an interleaved fashion.

![Diagram of sequential consistency]

But...
- No multicore processor implements SC.
- Compiler optimizations invalidate SC.
The illusion of sequential consistency

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.
Weak consistency

Hardware provides weak consistency.

- Weak memory models \(\sim\) semantics of shared memory.
- Every hardware architecture has its own WMM: x86-TSO, ARM, Power, Itanium.

**x86-TSO model (2010)**

```
CPU -> write -> Memory
read ^...^ read
```

**ARMv8 model (2016)**

```
CPU ^...^ CPU
Memory ^...^ Memory
```

6
Weak consistency examples

**Store buffering (SB)**

Initially, \( x = y = 0 \)

\[
\begin{align*}
    x &:= 1; & y &:= 1; \\
    a &:= y \; // 0 & b &:= x \; // 0
\end{align*}
\]

**Load buffering (LB)**

Initially, \( x = y = 0 \)

\[
\begin{align*}
    a &:= y \; // 1 & b &:= x \; // 1 \\
    x &:= 1 & y &:= 1
\end{align*}
\]
There is more to WMC than just reorderings

Independent reads of independent writes (IRIW)

Initially, \( x = y = 0 \)

\[
\begin{align*}
x &:= 1 & & a := x; & & c := y; & & y := 1 \text{ } / / 1 \\
& & lwsync; & & lwsync; & & \\
& & b := y & & d := x & & / / 0
\end{align*}
\]

- Thread II and III can observe the \( x := 1 \) and \( y := 1 \) writes happen in different orders.
- Because of the \texttt{lwsync} fences, no reorderings are possible!
WMC is not just about hardware

C/C++/Java/…

GCC

ELVM

x86  Power  ARM
Quiz. Should these transformations be allowed?

1. CSE over acquiring a lock:

\[
\begin{align*}
a &= x; \\
lock(); &\quad \mapsto \\
b &= x; \\
\end{align*}
\]

2. Load hoisting:

\[
\begin{align*}
\text{if (c)} \\
a &= x; &\quad \mapsto \\
t &= x; \\
a &= c ? t : a;
\end{align*}
\]

[x is a global variable; a, b, c are local; t is a fresh temporary.]
Allowing both is clearly wrong!

Consider the transformation sequence:

\[
\begin{align*}
&\text{if } (c) \quad t = x; \quad t = x; \\
&\quad a = x; \quad \text{hoist} \quad a = c ? t : a; \quad \text{CSE} \quad a = c ? t : a; \\
&\quad \text{lock();} \quad \text{lock();} \quad \text{lock();} \\
&\quad b = x; \quad b = x; \quad b = t;
\end{align*}
\]

When \( c \) is false, \( x \) is moved out of the critical region!

So we have to forbid one transformation.

- C11 forbids load hoisting, allows CSE over lock().
- LLVM allows load hoisting, forbids CSE over lock().
Weak consistency in “real life”

▶ Messages may be delayed.

\[ \begin{align*}
    MsgX & := 1; \\
    a & := MsgY; \quad \text{// 0} \\
    MsgY & := 1; \\
    b & := MsgX; \quad \text{// 0}
\end{align*} \]

▶ Messages may be sent/received out of order.

\[ \begin{align*}
    Email & := 1; \\
    Sms & := 1; \\
    a & := Sms; \quad \text{// 1} \\
    b & := Email; \quad \text{// 0}
\end{align*} \]
**Embracing weak consistency**

*Weak consistency* is not a threat, but an opportunity.
- Can lead to more scalable concurrent algorithms.
- Several open research problems.
  - What is a good memory model?

*Reasoning under WMC is often easier than under SC.*
- Avoid thinking about thread interleavings.
- Many/most concurrent algorithms do not need SC!
- Positive *vs* negative knowledge.
Memory model definitions

- Operational memory models
- Axiomatic/declarative memory models
- Promising semantics

WMM metatheory

- Relating memory models
- Correctness of compilation and program transformations
- Programming guarantees: the DRF theorem

Verification techniques for WMM

- Program logics (relaxed separation logic, OGRA)
- Model checking