Memory Consistency Models: Convergence At Last!

Sarita Adve

Department of Computer Science

University of Illinois at Urbana-Champaign

sadve@cs.uiuc.edu

Acks:

Co-authors: *Mark Hill*, Kourosh Gharachorloo, Anoop Gupta, John Hennessy, Vijay Pai, Partha Ranganathan, Jeremy Manson, Bill Pugh, Hans Boehm

Others: Doug Lea, Bratin Saha, Herb Sutter, and many more

What is a Memory Consistency Model?

- **Memory model defines what values a read can return**

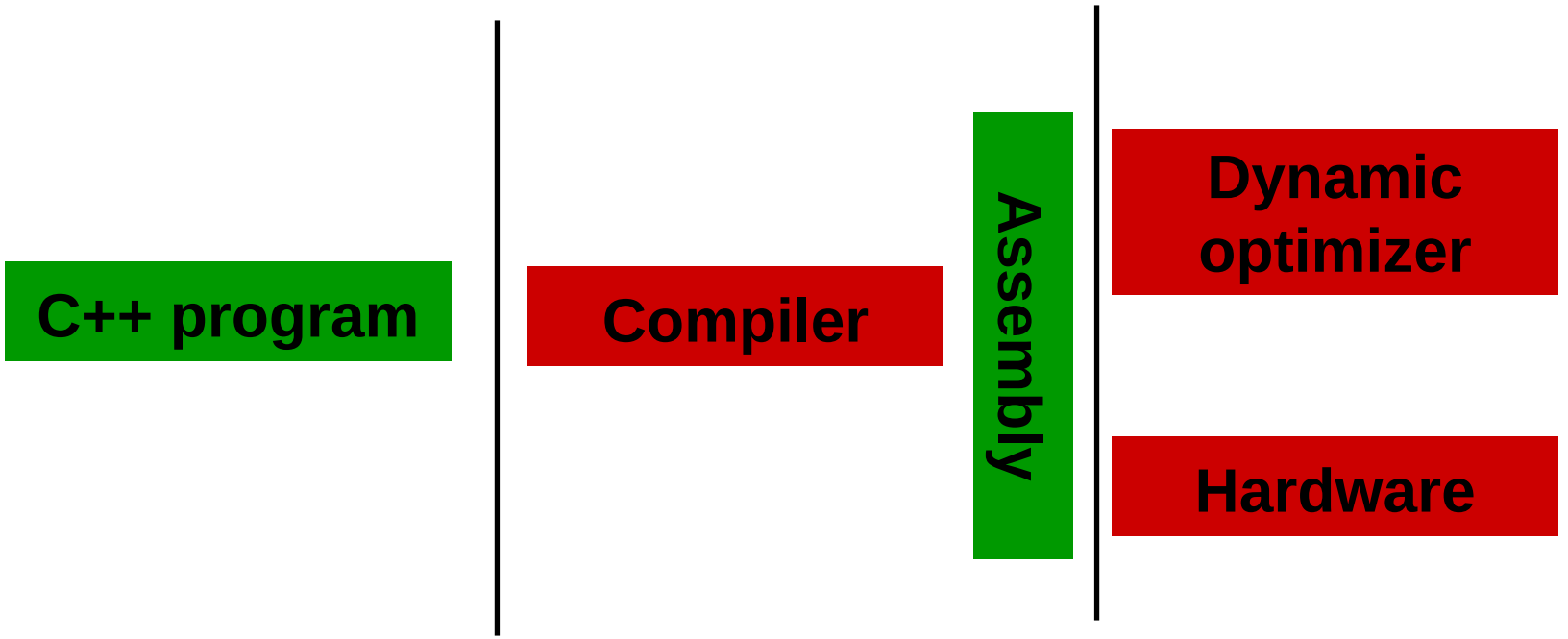
Initially A=B=C=Flag=0

Thread 1	*Thread 2*
A = 26	while (Flag != 1) {;}
B = 90	r1 = B ← ~~90~~
…	r2 = A ← ~~26~~
Flag = 1	…

0

What is a Memory Model?

- **Interface between program and transformers of program**
  - **Defines what values a read can return**



- **Language level model has implications for hardware**
- **Weakest system component exposed to programmer**

Desirable Properties of a Memory Model

- **3 Ps**
  - **Programmability**
  - **Performance**
  - **Portability**

- **Challenge: hard to satisfy all 3 Ps**
  - **Late 1980's - 90's: Largely driven by hardware**
    - **Lots of models, little consensus**
  - **2000 onwards: Largely driven by languages/compilers**
    - **Consensus model for Java, C++, Microsoft native code**
    - **Most major hardware vendors on board**

This talk: Path to convergence – challenges, limitations, implications

Programmability – SC [Lamport79]

- **Programmability: Sequential consistency (SC) most intuitive**
  - **Accesses of a single thread in program order**
  - **All accesses in a total order or atomic**
- **But Performance?**
  - **Recent hardware techniques boost performance with SC**
  - **But compiler transformations still inhibited**
- **But Portability?**
  - **Almost all current hardware, compilers violate SC**

⇒ SC not practical yet, but…

Next Best Thing – SC for Almost Everyone

- **Parallel programming too hard even with SC**
  - **Programmers write well structured code**
  - **Explicit synchronization, no data races**

Thread 1	*Thread 2*
Lock(L)	Lock(L)
Read Data1	Read Data2
Write Data2	Write Data1
Write Data3	Read Data3
Read Data2	Write Data3
…	…
Unlock(L)	Unlock(L)

  - **SC for such programs much easier: can reorder data accesses**

⇒ **Data-race-free model**

  - **SC for data-race-free programs**
  - **No guarantees for programs with data races**

[AdveHill, Gharachorloo et al. 1990s]

Definition of a Data Race

- **Only need to define for SC executions ⇒ total order**
- **Two memory accesses form a race if**
  - **From different threads, to same location, at least one is a write**
  - **Occur one after another**

Thread 1	Thread 2
Write, A, 26	
Write, B, 90	
	Read, Flag, 0
Write, Flag, 1	
	Read, Flag, 1
	Read, B, 90
	Read, A, 26

- **A race with a data access is a data race**
- **Data-race-free-program = No data race in any SC execution**

Data-Race-Free Model

Data-race-free model = SC for data-race-free programs

- **Does not preclude races for wait-free constructs, etc.**
  - **Requires races be explicitly identified as synch**
  - **E.g., use volatile variables in Java, atomics in C++**
- **Dekker's algorithm**

Initially Flag1 = Flag2 = 0

volatile Flag1, Flag2

Thread1	***Thread2***
Flag1 = 1	**Flag2 = 1**
if Flag2 == 0	**if Flag1 == 0**
//critical section	**//critical section**

SC prohibits both loads returning 0

Data-Race-Free Approach

- **Programmer's model: SC for data-race-free programs**
- **Programmability**
  - **Simplicity of SC, for data-race-free programs**
- **Performance**
  - **Specifies minimal constraints (for SC-centric view)**
- **Portability**
  - **Language must provide way to identify races**
  - **Hardware must provide way to preserve ordering on races**
  - **Compiler must translate correctly**

1990's in Practice

- **Hardware**
  - **Different vendors had different models – most non-SC**
    - **Alpha, Sun, x86, Itanium, IBM, AMD, HP, Convex, Cray, …**
  - **Various ordering guarantees + fences to impose other orders**
  - **Many ambiguities - due to complexity, by design(?), …**

- **High-level languages**
  - **Most shared-memory programming with Pthreads, OpenMP**
    - **Incomplete, ambiguous model specs**
    - **Memory model property of language, not library**
  - **Java – commercially successful language with threads**
    - **Chapter 17 of Java language spec described memory model**
    - **But hard to interpret, badly broken**

2000 – 2004: Java Memory Model

- **~ 2000: Bill Pugh publicized fatal flaws in Java memory model**
- **Lobbied Sun to form expert group to revise Java model**
- **Open process via mailing list with diverse subscribers**
  - **Took 5 years of intense, spirited debates**
  - **Many competing models**
  - **Final consensus model approved in 2005 for Java 5.0**

[MansonPughAdve POPL 2005]

Java Memory Model - Highlights

- **Quick agreement that SC for data-race-free was required**
- **Missing piece: Semantics for programs with data races**
  - **Java cannot have undefined semantics for ANY program**
  - **Must ensure safety/security guarantees of language**
- **Goal: minimal semantics for races to satisfy security/safety**
  - **Problem: safety/security issues for multithreading very vague**
- **Final model based on consensus, but complex [POPL05]**
  - **But programmers can program with "SC for data-race-free"**
  - **Can use races, but declare them volatile**

2005 - :C++, Microsoft Prism, Multicore

- **~ 2005: Hans Boehm started effort for C++ concurrency model**
  - **Prior status: no threads in C++, most concurrency w/ Pthreads**
- **Microsoft concurrently started its own internal effort**
- **C++ easier than Java because it is unsafe**
  - **Data-race-free is plausible model**
- **BUT**
  - **Multicore ⇒ New h/w optimizations, h/w vendors cared more**
  - **Pthreads has larger set of synchronization techniques**
  - **Can we really get away with no semantics for data races?**

Hardware Implications of Data-Race-Free

- **Synchronization (volatiles/atomics) must appear SC**
  - **Each thread's synch must appear in program order**

synch Flag1, Flag2

T1	T2
Flag1 = 1	Flag2 = 1
→ Fence	Fence
if Flag2 == 0	if Flag1 == 0
critical section	critical section

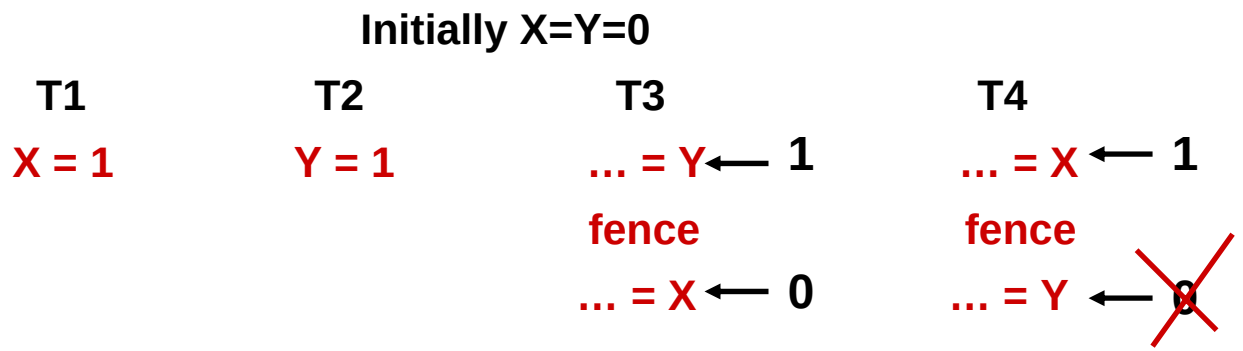
SC ⇒ both reads cannot return 0

Requires efficient fences between synch stores/loads

  - **All synch writes must appear in a total order (atomic)**

Implications of Atomic Synch Writes

Independent reads, independent writes (IRIW):



SC ⇒ no thread sees new value until old copies invalidated

- **Shared caches w/ hyperthreading/multicore make this harder**
- **Programmers don't usually use IRIW**
- **Why pay cost for SC in h/w if not useful to s/w?**

Implications of Atomic Synch Writes

2006: Pressure to change Java/C++ to remove SC volatiles

- **But what is alternative?**
  - **Must allow non-SC outcome for IRIW**
  - **But must be teachable to undergrads**
- **Showed such an alternative (probably) does not exist**
  - **IRIW style optimizations give non-intuitive results for codes that programmers do care about**
  - **Violates composability of cache coherence and causality**
  - **Unacceptable for Joe**

C++ Compromise

- **Default C++ model is data-race-free**
- **AMD, Intel, … on board**
  - **Provide way to give SC synchs**
  - **Synch writes replaced with xchg**
- **But**
  - **Some IBM systems need expensive fence for SC IRIW**
  - **Some programmers really want more flexibility**
    - **C++ specifies low-level atomics only for experts**
    - **Complicates spec, but only for experts**

Status So Far

- **Convergence to “SC for data-race-free programs” as baseline**
- **For safe languages, minimal semantics for data races**
  - **Implications for compilers**
  - **NOT for programmers**
- **Hardware: fences for program order, atomicity for some writes**
- **Compiler: translate synch (volatiles/atomics) to correct fences, atomic writes**
- **For super-duper-expert programmers, non-SC flavors of synch**
  - **Don’t teach this in undergrad class**

Lessons Learned (1 of 2)

- **Specifying semantics for programs with data races is HARD**
  - **But "no semantics for data races" also has problems**
    - **Debugging**
    - **Source-to-source compilers cannot introduce data races**

  $\Rightarrow$ **Need languages that banish data races**

- **But should programmers have to reason about reads and writes?**
  - **Need higher level *programming* models**
    - **Must inherently be "race-free"**
    - **Transactions are only part of the story**

Lessons Learned (2 of 2)

- **Simple optimizations can have unintended consequences**
  - **A little extra performance is not worth the larger increase in programming complexity**
- **DRF/Java models not prescriptive for hardware, compilers**
  - **Hard to verify when hardware, compilers obey the models**
  - **Abstractions for describing/verifying hardware/compilers**
- **Affecting a standard is different from writing a paper**
  - **It helped to have Microsoft on our side…**
- **Stick with it!**
  - **Especially for work crossing boundaries**