

# A Solver-Aided DSL for Understanding Weak-Memory Models

Gowtham Kaki

Final Project of CS590  
Fall, 2014



# Multi-Cores ...

... and many-cores have proliferated

Shared-memory concurrent programming remains difficult even under the assumption of sequential consistency (SC)

Initially X=Y=0	
Red Thread	Blue Thread
X = 1;	Y = 1;
r1 = Y;	r2 = X;

What values can  
r1 and r2 take  
assuming SC?

## Multi-Cores ...

... and many-cores have proliferated

Shared-memory concurrent programming remains difficult even under the assumption of sequential consistency (SC)

Initially X=Y=0	
Red Thread	Blue Thread
X = 1;	Y = 1;
r1 = Y;	r2 = X;

What values can  
r1 and r2 take  
assuming SC?

But, supporting SC by default is an expensive proposition.

- Disables compiler optimizations (eg: common sub-exp. elim.)
- Requires hardware to support cache-coherence



C++ and Java memory models do not offer SC by default.

# Weaker consistency models

C++ and Java memory models do not offer SC by default.

Reasoning under weak memory models is difficult!

For eg:

Initially X=Y=0	
Red Thread	Blue Thread
X = 1;	Y = 1;
r1 = Y;	r2 = X;

Not possible under SC

is  $r1 = r2 = 0$  possible?

But, possible in C++ & Java memory models!

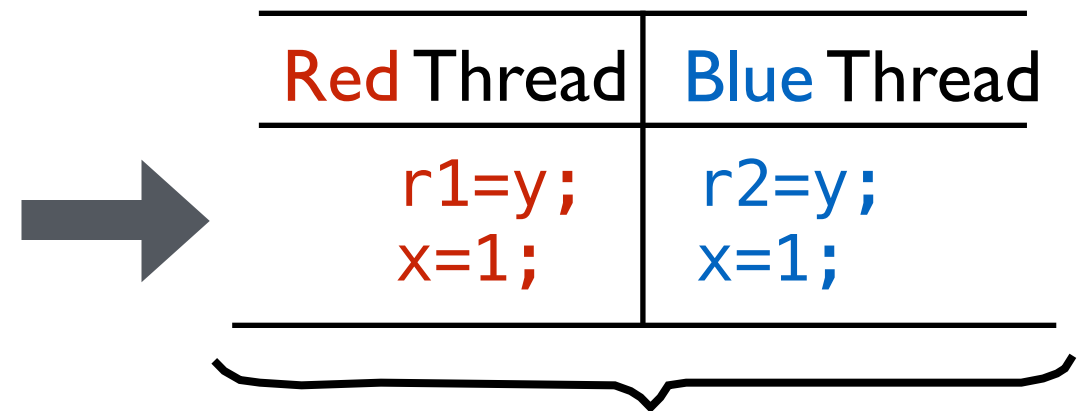
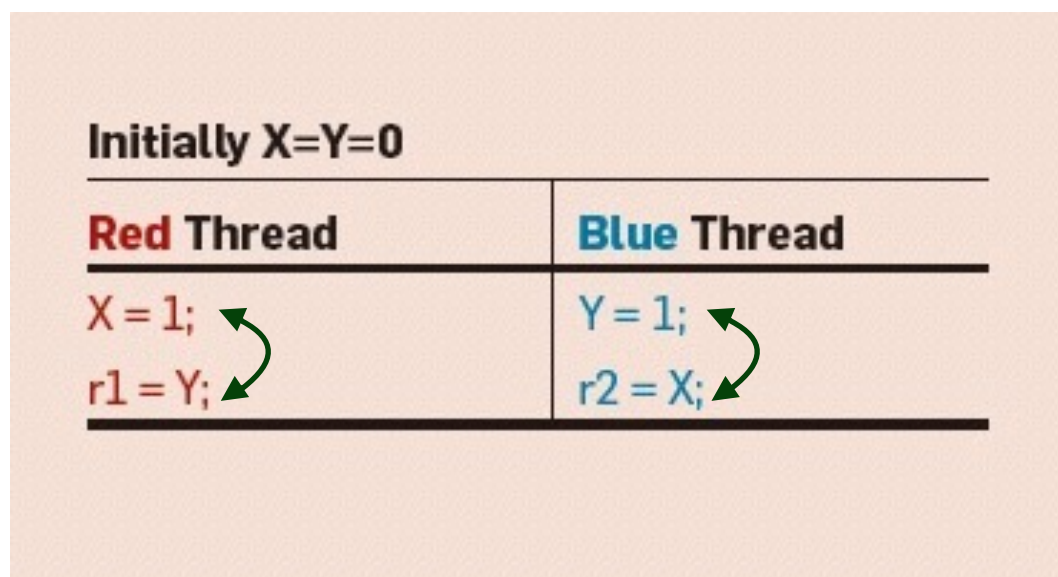
- Hardware optimizations - write buffer.  
writes applied to write buffer and lazily committed to main memory, whereas reads are always served by the main memory.
- Compiler optimizations - hoisting reads.

Can be understood  
as instruction re-orderings  
violating program order!

# Weaker consistency models

C++ and Java memory models do not offer SC by default.

Reasoning under weak memory models is difficult!



Now,  $r1 = r2 = 0$  is possible under SC

- Hardware optimizations - write buffer.  
writes applied to write buffer and lazily committed to main memory, whereas reads are always served by the main memory.
- Compiler optimizations - hoisting reads.

Can be understood  
as instruction re-orderings  
violating program order!

# Observation

Various compiler and hardware optimizations can be described in terms of\*:

- A. How does a compiler optimization reorder instructions within a thread
- B. Does a memory operation performed by a thread becomes visible at rest of the threads atomically?
- C. Does two memory operations ordered by program order in a thread become visible at rest of the threads in the same order?

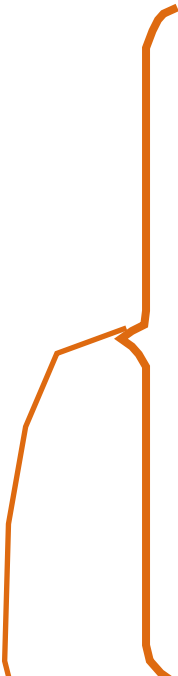
For eg, a system with simple store-buffer hardware optimization, but no compiler optimizations can be described by answering above qns as:

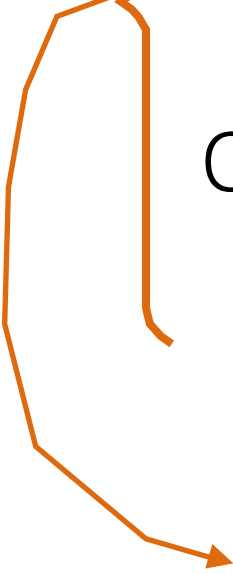
- A. No re-orderings
- B. No
- C. Yes

\*This view is also supported by observations made in [S Adve, and H-J Boehm, Shared Memory Consistency Models - A Tutorial, 1995].

# Insight

Various compiler and hardware optimizations can be described in terms of:

- 
- A. How does a compiler optimization reorder instructions within a thread
  - B. Does a memory operation performed by a thread becomes visible at rest of the threads atomically?
  - C. Does two memory operations ordered by program order in a thread become visible at rest of the threads in the same order?



The answers to these questions can be encoded as a function that takes an abstract representation of shared-memory program as list of list of LOADs and STOREs, and performs all possible re-orderings non-deterministically.

# Example

Initially X=Y=0	
Red Thread	Blue Thread
X = 1;	Y = 1;
r1 = Y;	r2 = X;

Expected behaviors

$(r1, r2) \in \{(0, 1), (1, 0), (1, 1)\}$

List of lists representation

$\{ [ \text{STORE}(x, 1); \text{LOAD}(r1, y), \text{STORE}(y, 1); \text{LOAD}(r2, x) ] \}$