

# Memory Consistency Models

Raghesh A (CS12D015)

PACE Lab, Department of CSE, IIT Madras

January 3, 2015

## Why ...?

- Multithreaded programs run on either
  - a Uniprocessor

## Why ...?

- Multithreaded programs run on either
  - a Uniprocessor OR
  - Multicores.

## Why ...?

- Multithreaded programs run on either
  - a Uniprocessor OR
  - Multicores.
- One simple way to argue about the correctness of multithreaded programs is by looking at the output produced.

## Why ...?

- Multithreaded programs run on either
  - a Uniprocessor OR
  - Multicores.
- One simple way to argue about the correctness of multithreaded programs is by looking at the output produced.
- Multiple outputs may be acceptable.

## Why ...?

- Multithreaded programs run on either
  - a Uniprocessor OR
  - Multicores.
- One simple way to argue about the correctness of multithreaded programs is by looking at the output produced.
- Multiple outputs may be acceptable.
- The indeterministic nature of multithreaded programs makes programmer's life difficult to ensure correctness.

## Why ...?

- Multithreaded programs run on either
  - a Uniprocessor OR
  - Multicores.
- One simple way to argue about the correctness of multithreaded programs is by looking at the output produced.
- Multiple outputs may be acceptable.
- The indeterministic nature of multithreaded programs makes programmer's life difficult to ensure correctness.
  - One common approach is to apply synchronization.
  - But the degree of synchronization must be minimal.
  - Basically synchronization controls the order of read and write to shared memory locations.
  - Without sychronization the order of read and writes may be altered either by compiler (optimizations) or by hardware for better performance.

## Why ...?

- Multithreaded programs run on either
  - a Uniprocessor OR
  - Multicores.
- One simple way to argue about the correctness of multithreaded programs is by looking at the output produced.
- Multiple outputs may be acceptable.
- The indeterministic nature of multithreaded programs makes programmer's life difficult to ensure correctness.
  - One common approach is to apply synchronization.
  - But the degree of synchronization must be minimal.
  - Basically synchronization controls the order of read and write to shared memory locations.
  - Without synchronization the order of read and writes may be altered either by compiler (optimizations) or by hardware for better performance.
- The objective.
  - Less compromise on performance - With less degree of synchronization shall we get acceptable output ????
  - Arguing about the correctness of the program.



# Memory Model

- A formal specification of how the memory system will appear to the programmer.
- Eliminating gap between expected behaviour (by the programmer) and the actual behaviour.
- Essentially, a **Memory Model** defines the legitimate orderings of read and write to memory locations.

# Memory Model

- A formal specification of how the memory system will appear to the programmer.
- Eliminating gap between expected behaviour (by the programmer) and the actual behaviour.
- Essentially, a **Memory Model** defines the legitimate orderings of read and write to memory locations.
  - A strict memory model - Easy to argue about correctness, but restricts optimizations (both at compiler and hardware level).
  - A relaxed memory model - More opportunities for optimizations, but not easy to argue correctness.

# Memory Model

- A formal specification of how the memory system will appear to the programmer.
- Eliminating gap between expected behaviour (by the programmer) and the actual behaviour.
- Essentially, a **Memory Model** defines the legitimate orderings of read and write to memory locations.
  - A strict memory model - Easy to argue about correctness, but restricts optimizations (both at compiler and hardware level).
  - A relaxed memory model - More opportunities for optimizations, but not easy to argue correctness.

## Sequential Consistency (SC)

- Uniprocessor - A "read" returns the last "write" in program order.
- Multiprocessor - A "read" may/may not return a value according to "writes" in program order.
  - Affected by cache coherency, use of write-buffers, register allocation, ...

## Sequential Consistency (SC)

- Uniprocessor - A "read" returns the last "write" in program order.
- Multiprocessor - A "read" may/may not return a value according to "writes" in program order.
  - Affected by cache coherency, use of write-buffers, register allocation, ...
- **Sequential consistency** - An intuitive extension of uniprocessor model.
  - **Program order**: Each core executes statements in program order.
  - Cores are switched in an arbitrary (indeterministic) manner.
  - **Atomicity**: A memory operation executes atomically w.r.t. other memory operations.

## Sequential Consistency (SC)

- Uniprocessor - A "read" returns the last "write" in program order.
- Multiprocessor - A "read" may/may not return a value according to "writes" in program order.
  - Affected by cache coherency, use of write-buffers, register allocation, ...
- **Sequential consistency** - An intuitive extension of uniprocessor model.
  - **Program order**: Each core executes statements in program order.
  - Cores are switched in an arbitrary (indeterministic) manner.
  - **Atomicity**: A memory operation executes atomically w.r.t. other memory operations.
- An Example [1]

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

## Sequential Consistency (SC)

- Uniprocessor - A "read" returns the last "write" in program order.
- Multiprocessor - A "read" may/may not return a value according to "writes" in program order.
  - Affected by cache coherency, use of write-buffers, register allocation, ...
- **Sequential consistency** - An intuitive extension of uniprocessor model.
  - **Program order**: Each core executes statements in program order.
  - Cores are switched in an arbitrary (indeterministic) manner.
  - **Atomicity**: A memory operation executes atomically w.r.t. other memory operations.
- An Example [1]

Initially  $X = Y = 0$

Red Thread	Blue Thread
$X = 1;$	$Y = 1;$
$r1 = Y;$	$r2 = X;$

- Some possible interleavings.

Execution 1	Execution 2	Execution 3
$X = 1;$	$Y = 1;$	$X = 1;$
$r1 = Y;$	$r2 = X;$	$Y = 1;$
$Y = 1;$	$X = 1;$	$r1 = Y;$
$r2 = X;$	$r1 = Y;$	$r2 = X;$
// $r1 == 0;$	// $r1 == 1;$	// $r1 == 1;$
// $r2 == 1;$	// $r2 == 0;$	// $r2 == 1;$

## Relaxed Consistency (RC)

- Sequential consistency is hard to realize.
- Too much restriction for hardware and compiler optimizations
- So we require **Relaxed Consistency Models**.
- Same program may exhibit more executions on a relaxed model than SC [2].
  - Let  $T_{\Pi}^Y$  be the set of executions on memory model Y. Then  $T_{\Pi}^Y \subset T_{\Pi}^{SC}$ .
  - TODO: Write formally from paper
  - Write the two steps to verify a problem in RC.



# Linearisability



Sarita V. Adve and Hans-J. Boehm.

Memory models: A case for rethinking parallel languages and hardware.

*Commun. ACM*, 53(8):90–101, August 2010.



Sebastian Burckhardt and Madanlal Musuvathi.

Effective program verification for relaxed memory models.

In *Proceedings of the 20th International Conference on Computer Aided Verification, CAV '08*, pages 107–120, Berlin, Heidelberg, 2008. Springer-Verlag.